

SUNDIALSTB v2.2.0, a MATLAB Interface to SUNDIALS

Radu Serban
*Center for Applied Scientific Computing
Lawrence Livermore National Laboratory*

May 12, 2006



UCRL-SM-212121

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This research was supported under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Contents

1	Introduction	1
1.1	Notes	1
1.2	Requirements	1
1.3	Installation/Setup	1
1.3.1	Choose ubication	1
1.3.2	Decompress and untar	2
1.3.3	Configuring MATLAB's startup	2
1.3.4	Compile MEX files	3
1.3.5	Try one of the sundialsTB examples	3
1.4	Links	3
2	MATLAB Interface to CVODES	4
2.1	Interface functions	5
2.2	Function types	35
3	MATLAB Interface to KINSOL	48
3.1	Interface functions	49
3.2	Function types	56
4	Supporting modules	63
4.1	NVECTOR functions	64
4.2	Parallel utilities	70
	References	80

1 Introduction

SUNDIALS [2], SUite of Nonlinear and DIfferential/ALgebraic equation Solvers, is a family of software tools for integration of ODE and DAE initial value problems and for the solution of nonlinear systems of equations. It consists of CVODE, IDA, and KINSOL, and variants of these with sensitivity analysis capabilities.

SUNDIALSTB is a collection of MATLAB functions which provide interfaces to the SUNDIALS solvers.

The core of each MATLAB interface in SUNDIALSTB is a single MEX file which interfaces to the various user-callable functions for that solver. However, this MEX file should not be called directly, but rather through the user-callable functions provided for each MATLAB interface.

A major design principle for SUNDIALSTB was to provide an interface that is, as much as possible, equally familiar to both SUNDIALS users and MATLAB users. Moreover, we tried to keep the number of user-callable functions to a minimum. For example, the CVODES MATLAB interface contains only 12 such functions, 2 of which relate to forward sensitivity analysis and 4 more interface solely to the adjoint sensitivity module in CVODES. A user who is only interested in integration of ODEs and not in sensitivity analysis therefore needs to call at most 6 functions. In tune with the MATLAB ODESET function, optional solver inputs in SUNDIALSTB are specified through a single function; e.g. `CvodeSetOptions` for CVODES (a similar function is used to specify optional inputs for forward sensitivity analysis). However, unlike the ODE solvers in MATLAB, we have kept the more flexible SUNDIALS model in which a separate “solve” function (`CVodeSolve` for CVODES) must be called to return the solution at a desired output time. Solver statistics, as well as optional outputs (such as solution and solution derivatives at additional times) can be obtained at any time with calls to separate functions (`CVodeGetStats` and `CVodeGet` for CVODES).

This document provides a complete documentation for the SUNDIALSTB functions. For additional details on the methods and underlying SUNDIALS software consult also the corresponding SUNDIALS user guides [3, 1].

1.1 Notes

The version numbers for the MATLAB interfaces correspond to those of the corresponding SUNDIALS solver with which the interface is compatible.

1.2 Requirements

Each interface module in SUNDIALSTB requires the appropriate version of the corresponding SUNDIALS solver. For parallel support, SUNDIALSTB depends on MPITB with LAM v > 7.1.1 (for MPI-2 spawning feature).

1.3 Installation/Setup

The following steps are required to install and setup SUNDIALSTB:

1.3.1 Choose ubication

1. SUNDIALSTB for all MATLAB users (not usual)

Assume MATLAB is installed under \$MATLAB=/usr/local/matlab7. Place sundialsTB where toolboxes are usually stored:

```
$ cd /usr/local/matlab7/toolbox
```

2. SUNDIALSTB for just one user (usual configuration)

Place sundialsTB in your ”matlab” working subdir:

```
$ cd ~/matlab
```

1.3.2 Decompress and untar

```
$ tar zxvf <wherever>/sundialsTB.tar.gz
```

or

```
$ cp <wherever>/sundialsTB.tar.gz
$ gunzip sundialsTB.tar.gz
$ tar xvf sundialsTB.tar
$ rm sundialsTB.tar
```

Now there is a /matlab/sundialsTB (or \$MATLAB/toolbox/sundialsTB) subdirectory.

1.3.3 Configuring MATLAB's startup

SUNDIALSTB comes with a startup_STB.m file in the top subdirectory

1. SUNDIALSTB for all MATLAB users (not usual)

Assume MATLAB is installed under \$MATLAB=/usr/local/matlab7. Add SUNDIALSTB startup to the system-wide startup file:

```
$ cd \$MATLAB/toolbox/local
$ ln -s ../sundialsTB/startup_STB.m
```

and add these lines to your original local startup.m

```
% SUNDIALS Toolbox startup M-file, if it exists.
if exist('startup_STB','file')
    startup_STB
end
```

2. SUNDIALSTB for just one user (usual configuration)

Assume you do not need to keep any previously existing startup.m

```
$ cd ~/matlab
$ ln -s sundialsTB/startup_STB.m startup.m
```

If you already had a startup.m, use the method described above, first linking startup_STB.m to the destination subdir and then editing /matlab/startup.m to run startup_STB.m

3. Since symbolic links do not work well with Matlab under Windows, a different alternative of setting-up the startup is as follows:

Copy startup_STB next to MATLAB's startup.m and add these lines to startup.m

```
% SUNDIALS Toolbox startup M-file
startup('path_to_sundialsTB')
```

using the optional argument of startup_STB to explicitly specify the location of sundialsTB.

1.3.4 Compile MEX files

To facilitate the compilation of SUNDIALSTB on platforms that do not have a make system, we rely on MATLAB's mex command. Compilation of sundialsTB is done by running from under MATLAB the install_STB.m script which is present in the sundialsTB top directory.

1. If you have not already done so, download and unpack SUNDIALS. Assume SUNDIALS is now located in /path/to/sundials
2. Launch matlab in sundialsTB:

```
$ cd $MATLAB/toolbox/sundialsTB  
$ matlab
```

or

```
$ cd ~/matlab/sundialsTB  
$ matlab
```

3. Run the install_STB matlab script. You will be asked for the location of the SUNDIALS source tree. Input /path/to/sundials.

Note that parallel support will be compiled into the MEX files only if ALL of the following conditions are met:

- \$LAMHOME is defined
- \$MPITB_ROOT is defined
- /path/to/sundials/nvec_par exists

1.3.5 Try one of the sundialsTB examples

If everything went fine, you should now be able to try one of the CVODES or KINSOL examples (in matlab, type 'help cvodes' or 'help kinsol' to see a list of all examples available).

1. cd to the CVODES serial example directory

```
$ cd $MATLAB/toolbox/sundialsTB/cvodes/examples_ser
```

or

```
$ cd ~/matlab/sundialsTB/cvodes/examples_ser
```

2. Launch matlab and execute cvdx

1.4 Links

The required software packages can be obtained from the following addresses.

SUNDIALS <http://www.llnl.gov/CASC/sundials>
MPITB http://atc.ugr.es/javier-bin/mpitb_eng
LAM <http://www.lam-mpi.org/>

2 MATLAB Interface to CVODES

The MATLAB interface to CVODES provides access to all functionality of the CVODES solver, including IVP simulation and sensitivity analysis (both forward and adjoint).

The interface consists of 9 user-callable functions. The user must provide several required and optional user-supplied functions which define the problem to be solved. The user-callable functions and the types of user-supplied functions are listed in Table 1 and fully documented later in this section. For more in depth details, consult also the CVODES user guide [3].

To illustrate the use of the CVODES MATLAB interface, several example problems are provided with SUNDIALSTB, both for serial and parallel computations. Most of them are MATLAB translations of example problems provided with CVODES.

Table 1: CVODES MATLAB interface functions

Functions	CVodeSetOptions CVodeSetFSAOptions CVodeMalloc CVodeSensMalloc CVadjMalloc CVodeMallocB CVode CVodeB CVodeGetStats CVodeGetStatsB CVodeGet CVodeFree CVodeMonitor	creates an options structure for CVODES. creates an options structure for FSA with CVODES. allocates and initializes memory for CVODES. allocates and initializes memory for FSA with CVODES. allocates and initializes memory for ASA with CVODES. allocates and initializes backward memory for CVODES. integrates the ODE. integrates the backward ODE. returns statistics for the CVODES solver. returns statistics for the backward CVODES solver. extracts data from CVODES memory. deallocates memory for the CVODES solver. sample monitoring function.
Function types	CVRhsFn CVRootFn CVQuadRhsFn CVDenseJacFn CVBandJacFn CVJacTimesVecFn CVPrecSetupFn CVPrecSolveFn CVGlocalFn CVGcommFn CVSensRhsFn CVMonitorFn	RHS function root-finding function quadrature RHS function dense Jacobian function banded Jacobian function Jacobian times vector function preconditioner setup function preconditioner solve function RHS approximation function (BBDPre) communication function (BBDPre) sensitivity RHS function monitoring function

2.1 Interface functions

CVodeSetOptions

PURPOSE

CVodeSetOptions creates an options structure for CVODES.

SYNOPSIS

```
function options = CVodeSetOptions(varargin)
```

DESCRIPTION

CVodeSetOptions creates an options structure for CVODES.

Usage: OPTIONS = CVodeSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...)
OPTIONS = CVodeSetOptions(OLDOPTIONS,'NAME1',VALUE1,...)
OPTIONS = CVodeSetOptions(OLDOPTIONS,NEWOPTIONS)

OPTIONS = CVodeSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates a CVODES options structure OPTIONS in which the named properties have the specified values. Any unspecified properties have default values. It is sufficient to type only the leading characters that uniquely identify the property. Case is ignored for property names.

OPTIONS = CVodeSetOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an existing options structure OLDOPTIONS.

OPTIONS = CVodeSetOptions(OLDOPTIONS,NEWOPTIONS) combines an existing options structure OLDOPTIONS with a new options structure NEWOPTIONS. Any new properties overwrite corresponding old properties.

CVodeSetOptions with no input arguments displays all property names and their possible values.

CVodeSetOptions properties
(See also the CVODES User Guide)

LMM - Linear Multistep Method ['Adams' | 'BDF']

This property specifies whether the Adams method is to be used instead of the default Backward Differentiation Formulas (BDF) method.

The Adams method is recommended for non-stiff problems, while BDF is recommended for stiff problems.

NonlinearSolver - Type of nonlinear solver used [Functional | Newton]

The 'Functional' nonlinear solver is best suited for non-stiff problems, in conjunction with the 'Adams' linear multistep method, while 'Newton' is better suited for stiff problems, using the 'BDF' method.

RelTol - Relative tolerance [positive scalar | 1e-4]

RelTol defaults to 1e-4 and is applied to all components of the solution vector. See AbsTol.

AbsTol - Absolute tolerance [positive scalar or vector | 1e-6]

The relative and absolute tolerances define a vector of error weights with components

```

ewt(i) = 1/(RelTol*|y(i)| + AbsTol)    if AbsTol is a scalar
ewt(i) = 1/(RelTol*|y(i)| + AbsTol(i)) if AbsTol is a vector

```

This vector is used in all error and convergence tests, which use a weighted RMS norm on all error-like vectors v:

```

WRMSnorm(v) = sqrt( (1/N) sum(i=1..N) (v(i)*ewt(i))^2 ),
where N is the problem dimension.

```

MaxNumSteps - Maximum number of steps [positive integer | 500]
CVode will return with an error after taking MaxNumSteps internal steps in its attempt to reach the next output time.

InitialStep - Suggested initial stepsize [positive scalar]
By default, CVode estimates an initial stepsize h0 at the initial time t0 as the solution of

```

WRMSnorm(h0^2 ydd / 2) = 1

```

where ydd is an estimated second derivative of y(t0).

MaxStep - Maximum stepsize [positive scalar | inf]
Defines an upper bound on the integration step size.

MinStep - Minimum stepsize [positive scalar | 0.0]
Defines a lower bound on the integration step size.

MaxOrder - Maximum method order [1-12 for Adams, 1-5 for BDF | 5]
Defines an upper bound on the linear multistep method order.

StopTime - Stopping time [scalar]
Defines a value for the independent variable past which the solution is not to proceed.

RootsFn - Rootfinding function [function]
To detect events (roots of functions), set this property to the event function. See CVRootFn.

NumRoots - Number of root functions [integer | 0]
Set NumRoots to the number of functions for which roots are monitored.
If NumRoots is 0, rootfinding is disabled.

StabilityLimDet - Stability limit detection algorithm [on | off]
Flag used to turn on or off the stability limit detection algorithm within CVODES. This property can be used only with the BDF method.
In this case, if the order is 3 or greater and if the stability limit is detected, the method order is reduced.

LinearSolver - Linear solver type [Dense|Diag|Band|GMRES|BiCGStab|TFQMR]
Specifies the type of linear solver to be used for the Newton nonlinear solver (see NonlinearSolver). Valid choices are: Dense (direct, dense Jacobian), Band (direct, banded Jacobian), Diag (direct, diagonal Jacobian), GMRES (iterative, scaled preconditioned GMRES), BiCGStab (iterative, scaled preconditioned stabilized BiCG), TFQMR (iterative, scaled transpose-free QMR). The GMRES, BiCGStab, and TFQMR are matrix-free linear solvers.

JacobianFn - Jacobian function [function]
This property is overloaded. Set this value to a function that returns Jacobian information consistent with the linear solver used (see Linsolver). If not specified, CVODES uses difference quotient approximations.
For the Dense linear solver, JacobianFn must be of type CVDenseJacFn and must return a dense Jacobian matrix. For the Band linear solver, JacobianFn must be of type CVBandJacFn and must return a banded Jacobian matrix.
For the iterative linear solvers, GMRES, BiCGStab, and TFQMR, JacobianFn must be of type CVJacTimesVecFn and must return a Jacobian-vector product. This property is not used for the Diag linear solver.

KrylovMaxDim - Maximum number of Krylov subspace vectors [integer | 5]
 Specifies the maximum number of vectors in the Krylov subspace. This property is used only if an iterative linear solver, GMRES, BiCGStab, or TFQMR is used (see LinSolver).

GramSchmidtType - Gram-Schmidt orthogonalization [Classical | Modified]
 Specifies the type of Gram-Schmidt orthogonalization (classical or modified). This property is used only if the GMRES linear solver is used (see LinSolver).

PrecType - Preconditioner type [Left | Right | Both | None]
 Specifies the type of user preconditioning to be done if an iterative linear solver, GMRES, BiCGStab, or TFQMR is used (see LinSolver). PrecType must be one of the following: 'None', 'Left', 'Right', or 'Both', corresponding to no preconditioning, left preconditioning only, right preconditioning only, and both left and right preconditioning, respectively.

PrecModule - Preconditioner module [BandPre | BBDPre | UserDefined]
 If PrecModule = 'UserDefined', then the user must provide at least a preconditioner solve function (see PrecSolveFn)
 CVODES provides the following two general-purpose preconditioner modules:
 BandPre provide a band matrix preconditioner based on difference quotients of the ODE right-hand side function. The user must specify the lower and upper half-bandwidths through the properties LowerBwidth and UpperBwidth, respectively.
 BBDPre can be only used with parallel vectors. It provide a preconditioner matrix that is block-diagonal with banded blocks. The blocking corresponds to the distribution of the dependent variable vector y among the processors. Each preconditioner block is generated from the Jacobian of the local part (on the current processor) of a given function $g(t,y)$ approximating $f(t,y)$ (see GlocalFn). The blocks are generated by a difference quotient scheme on each processor independently. This scheme utilizes an assumed banded structure with given half-bandwidths, mldq and mudq (specified through LowerBwidthDQ and UpperBwidthDQ, respectively). However, the banded Jacobian block kept by the scheme has half-bandwidths ml and mu (specified through LowerBwidth and UpperBwidth), which may be smaller.

PrecSetupFn - Preconditioner setup function [function]
 If PrecType is not 'None', PrecSetupFn specifies an optional function which, together with PrecSolve, defines left and right preconditioner matrices (either of which can be trivial), such that the product $P1*P2$ is an approximation to the Newton matrix. PrecSetupFn must be of type CVPrecSetupFn.

PrecSolveFn - Preconditioner solve function [function]
 If PrecType is not 'None', PrecSolveFn specifies a required function which must solve a linear system $Pz = r$, for given r . PrecSolveFn must be of type CVPrecSolveFn.

GlocalFn - Local right-hand side approximation funciton for BBDPre [function]
 If PrecModule is BBDPre, GlocalFn specifies a required function that evaluates a local approximation to the ODE right-hand side. GlocalFn must be of type CVGlocFn.

GcommFn - Inter-process communication function for BBDPre [function]
 If PrecModule is BBDPre, GcommFn specifies an optional function to perform any inter-process communication required for the evaluation of GlocalFn. GcommFn must be of type CVGcommFn.

LowerBwidth - Jacobian/preconditioner lower bandwidth [integer | 0]
 This property is overloaded. If the Band linear solver is used (see LinSolver), it specifies the lower half-bandwidth of the band Jacobian approximation.
 If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used (see LinSolver) and if the BBDPre preconditioner module in CVODES is used

(see `PrecModule`), it specifies the lower half-bandwidth of the retained banded approximation of the local Jacobian block. If the `BandPre` preconditioner module (see `PrecModule`) is used, it specifies the lower half-bandwidth of the band preconditioner matrix. `LowerBwidth` defaults to 0 (no sub-diagonals).

`UpperBwidth` - Jacobian/preconditioner upper bandwidth [integer | 0]
This property is overloaded. If the `Band` linear solver is used (see `LinSolver`), it specifies the upper half-bandwidth of the band Jacobian approximation. If one of the three iterative linear solvers, `GMRES`, `BiCGStab`, or `TFQMR` is used (see `LinSolver`) and if the `BBDPre` preconditioner module in `CVODES` is used (see `PrecModule`), it specifies the upper half-bandwidth of the retained banded approximation of the local Jacobian block. If the `BandPre` preconditioner module (see `PrecModule`) is used, it specifies the upper half-bandwidth of the band preconditioner matrix. `UpperBwidth` defaults to 0 (no super-diagonals).

`LowerBwidthDQ` - `BBDPre` preconditioner DQ lower bandwidth [integer | 0]
Specifies the lower half-bandwidth used in the difference-quotient Jacobian approximation for the `BBDPre` preconditioner (see `PrecModule`).

`UpperBwidthDQ` - `BBDPre` preconditioner DQ upper bandwidth [integer | 0]
Specifies the upper half-bandwidth used in the difference-quotient Jacobian approximation for the `BBDPre` preconditioner (see `PrecModule`).

`Quadratures` - Quadrature integration [on | off]
Enables or disables quadrature integration.

`QuadRhsFn` - Quadrature right-hand side function [function]
Specifies the user-supplied function to evaluate the integrand for quadrature computations. See `CVQuadRhsfn`.

`QuadInitCond` - Initial conditions for quadrature variables [vector]
Specifies the initial conditions for quadrature variables.

`QuadErrControl` - Error control strategy for quadrature variables [on | off]
Specifies whether quadrature variables are included in the error test.

`QuadRelTol` - Relative tolerance for quadrature variables [scalar 1e-4]
Specifies the relative tolerance for quadrature variables. This parameter is used only if `QuadErrCon=on`.

`QuadAbsTol` - Absolute tolerance for quadrature variables [scalar or vector 1e-6]
Specifies the absolute tolerance for quadrature variables. This parameter is used only if `QuadErrCon=on`.

`ASANumDataPoints` - Number of data points for ASA [integer | 100]
Specifies the (maximum) number of integration steps between two consecutive check points.

`ASAInterpType` - Type of interpolation [Polynomial | Hermite]
Specifies the type of interpolation used for estimating the forward solution during the backward integration phase. At this time, the only option is 'Hermite', specifying cubic Hermite interpolation.

`MonitorFn` - User-provided monitoring function [function]
Specifies a function that is called after each successful integration step. This function must have type `CVMonitorFn`. A simple monitoring function, `CVodeMonitor` is provided with `CVODES`.

`MonitorData` - User-provided data for the monitoring function [struct]
Specifies a data structure that is passed to the `Monitor` function every time it is called.

See also

```
CVRootFn, CVQuadRhsFn  
CVDenseJacFn, CVBandJacFn, CVJacTimesVecFn  
CVPrecSetupFn, CVPrecSolveFn  
CVGlocalFn, CVGcommFn  
CVMonitorFn
```

CVodeSetFSAOptions

PURPOSE

CVodeSetFSAOptions creates an options structure for FSA with CVODES.

SYNOPSIS

```
function options = CVodeSetFSAOptions(varargin)
```

DESCRIPTION

CVodeSetFSAOptions creates an options structure for FSA with CVODES.

```
Usage: OPTIONS = CVodeSetFSAOptions('NAME1',VALUE1,'NAME2',VALUE2,...)  
       OPTIONS = CVodeSetFSAOptions(OLDOPTIONS,'NAME1',VALUE1,...)  
       OPTIONS = CVodeSetFSAOptions(OLDOPTIONS,NEWOPTIONS)
```

OPTIONS = CVodeSetFSAOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates a CVODES options structure OPTIONS in which the named properties have the specified values. Any unspecified properties have default values. It is sufficient to type only the leading characters that uniquely identify the property. Case is ignored for property names.

OPTIONS = CVodeSetFSAOptions(OLDOPTIONS,'NAME1',VALUE1,...) alters an existing options structure OLDOPTIONS.

OPTIONS = CVodeSetFSAOptions(OLDOPTIONS,NEWOPTIONS) combines an existing options structure OLDOPTIONS with a new options structure NEWOPTIONS. Any new properties overwrite corresponding old properties.

CVodeSetFSAOptions with no input arguments displays all property names and their possible values.

CVodeSetFSAOptions properties
(See also the CVODES User Guide)

FSAParamField - Problem parameters [string]

Specifies the name of the field in the user data structure (passed as an argument to CVodeMalloc) in which the nominal values of the problem parameters are stored. This property is used only if CVODES will use difference quotient approximations to the sensitivity right-hand sides (see SensRHS and SensRHStype).

FSAParamList - Parameters with respect to which FSA is performed [integer vector]
Specifies a list of Ns parameters with respect to which sensitivities are to be computed. This property is used only if CVODES will use difference-quotient approximations to the sensitivity right-hand sides (see SensRHS and SensRHStype). Its length must be Ns, consistent with the number of columns of FSAinitCond.

FSAParamScales - Order of magnitude for problem parameters [vector]
 Provides order of magnitude information for the parameters with respect to which sensitivities are computed. This information is used if CVODES approximates the sensitivity right-hand sides (see SensRHS) or if CVODES estimates integration tolerances for the sensitivity variables (see FSAReltol and FSAAbsTol).

FSAReltol - Relative tolerance for sensitivity variables [positive scalar]
 Specifies the scalar relative tolerance for the sensitivity variables.
 See FSAAbsTol.

FSAAbsTol - Absolute tolerance for sensitivity variables [row-vector or matrix]
 Specifies the absolute tolerance for sensitivity variables. FSAAbsTol must be either a row vector of dimension N_s , in which case each of its components is used as a scalar absolute tolerance for the corresponding sensitivity vector, or a $N \times N_s$ matrix, in which case each of its columns is used as a vector of absolute tolerances for the corresponding sensitivity vector.
 By default, CVODES estimates the integration tolerances for sensitivity variables, based on those for the states and on the order of magnitude information for the problem parameters specified through ParamScales.

FSAErrControl - Error control strategy for sensitivity variables [on | off]
 Specifies whether sensitivity variables are included in the error control test.
 Note that sensitivity variables are always included in the nonlinear system convergence test.

FSARhsFn - Sensitivity right-hand side function [function]
 Specifies a user-supplied function to evaluate the sensitivity right-hand sides. See SensRHStype. By default, CVODES uses an internal difference-quotient function to approximate the sensitivity right-hand sides.

FSADQparam - Parameter for the DQ approx. of the sensi. RHS [scalar | 0.0]
 Specifies the value which controls the selection of the difference-quotient scheme used in evaluating the sensitivity right-hand sides. This property is used only if CVODES will use difference-quotient approximations. The default value 0.0 indicates the use of the second-order centered directional derivative formula exclusively. Otherwise, the magnitude of FSADQparam and its sign (positive or negative) indicates whether this switching is done with regard to (centered or forward) finite differences, respectively.

See also
 CVSensRhsFn

CVodeMalloc

PURPOSE

CVodeMalloc allocates and initializes memory for CVODES.

SYNOPSIS

```
function [] = CVodeMalloc(fct,t0,y0,varargin)
```

DESCRIPTION

CVodeMalloc allocates and initializes memory for CVODES.

Usage: CVodeMalloc (ODEFUN, T0, Y0 [, OPTIONS [, DATA]])

ODEFUN is a function defining the ODE right-hand side: $y' = f(t,y)$.
 This function must return a vector containing the current
 value of the right-hand side.
T0 is the initial value of t .
Y0 is the initial condition vector $y(t_0)$.
OPTIONS is an (optional) set of integration options, created with
 the CVodeSetOptions function.
DATA is (optional) problem data passed unmodified to all
 user-provided functions when they are called. For example,
 $YD = \text{ODEFUN}(T,Y,DATA)$.

See also: `CVRhsFn`

`CVodeSensMalloc`

PURPOSE

`CVodeSensMalloc` allocates and initializes memory for FSA with CVODES.

SYNOPSIS

```
function [] = CVodeSensMalloc(Ns,meth,yS0,varargin)
```

DESCRIPTION

`CVodeSensMalloc` allocates and initializes memory for FSA with CVODES.

Usage: `CVodeSensMalloc (NS, METH, YS0 [, OPTIONS])`

NS is the number of parameters with respect to which sensitivities
 are desired
METHOD FSA solution method ['Simultaneous' | 'Staggered']
 Specifies the FSA method for treating the nonlinear system solution for
 sensitivity variables. In the simultaneous case, the nonlinear systems
 for states and all sensitivities are solved simultaneously. In the
 Staggered case, the nonlinear system for states is solved first and then
 the nonlinear systems for all sensitivities are solved at the same time.
YS0 Initial conditions for sensitivity variables.
 YS0 must be a matrix with N rows and Ns columns, where N is the problem
 dimension and Ns the number of sensitivity systems.
OPTIONS is an (optional) set of FSA options, created with
 the CVodeSetFSAOptions function.

`CVadjMalloc`

PURPOSE

`CVadjMalloc` allocates and initializes memory for ASA with CVODES.

SYNOPSIS

```
function [] = CVadjMalloc(steps, interp)
```

DESCRIPTION

`CVadjMalloc` allocates and initializes memory for ASA with CVODES.

Usage: `CVadjMalloc(STEPS, INTEPR)`

STEPS specifies the (maximum) number of integration steps between two consecutive check points.
INTERP Specifies the type of interpolation used for estimating the forward solution during the backward integration phase. **INTERP** should be 'Hermite', indicating cubic Hermite interpolation, or 'Polynomial', indicating variable order polynomial interpolation.

CCodeMallocB

PURPOSE

`CCodeMallocB` allocates and initializes backward memory for CVODES.

SYNOPSIS

```
function [] = CCodeMallocB(fctB,tB0,yB0,varargin)
```

DESCRIPTION

`CCodeMallocB` allocates and initializes backward memory for CVODES.

Usage: `CCodeMallocB (FCTB, TB0, YB0 [, OPTIONSB])`

FCTB is a function defining the adjoint ODE right-hand side.
This function must return a vector containing the current value of the adjoint ODE right-hand side.
TB0 is the final value of t.
YB0 is the final condition vector $y_B(tB0)$.
OPTIONSB is an (optional) set of integration options, created with the `CVodeSetOptions` function.

See also: `CVRhsFn`

CCode

PURPOSE

`CCode` integrates the ODE.

SYNOPSIS

```
function [status,t,y,varargout] = CCode(tout,itask)
```

DESCRIPTION

`CCode` integrates the ODE.

Usage: `[STATUS, T, Y] = CCode (TOUT, ITASK)`
`[STATUS, T, Y, YS] = CCode (TOUT, ITASK)`
`[STATUS, T, Y, YQ] = CCode (TOUT, ITASK)`
`[STATUS, T, Y, YQ, YS] = CCode (TOUT, ITASK)`

If ITASK is 'Normal', then the solver integrates from its current internal T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns Y(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step and returns in Y the solution at the new internal time. In this case, TOUT is used only during the first call to CVode to determine the direction of integration and the rough scale of the problem. In either case, the time reached by the solver is returned in T. The 'NormalTstop' and 'OneStepTstop' modes are similar to 'Normal' and 'OneStep', respectively, except that the integration never proceeds past the value tstop.

If quadratures were computed (see CVodeSetOptions), CVode will return their values at T in the vector YQ.

If sensitivity calculations were enabled (see CVodeSetOptions), CVode will return their values at T in the matrix YS.

On return, STATUS is one of the following:

- 0: CVode succeeded and no roots were found.
- 1: CVode succeeded and returned at tstop.
- 2: CVode succeeded, and found one or more roots.
- 1: Illegal attempt to call before CVodeMalloc
- 2: One of the inputs to CVode is illegal. This includes the situation when a component of the error weight vectors becomes < 0 during internal time-stepping.
- 4: The solver took mxstep internal steps but could not reach TOUT. The default value for mxstep is 500.
- 5: The solver could not satisfy the accuracy demanded by the user for some internal step.
- 6: Error test failures occurred too many times (MXNEF = 7) during one internal time step
or occurred with $|h| = hmin$.
- 7: Convergence test failures occurred too many times (MXNCF = 10) during one internal time step or occurred with $|h| = hmin$.
- 9: The linear solver's setup routine failed in an unrecoverable manner.
- 10: The linear solver's solve routine failed in an unrecoverable manner.

See also CVodeSetOptions, CVodeGetstats

CVodeB

PURPOSE

CVodeB integrates the backward ODE.

SYNOPSIS

```
function [status,t,yB,varargout] = CVodeB(tout,itask)
```

DESCRIPTION

CVodeB integrates the backward ODE.

```
Usage: [STATUS, T, YB] = CVodeB ( TOUT, ITASK )
       [STATUS, T, YB, YQB] = CVodeB ( TOUT, ITASK )
```

If ITASK is 'Normal', then the solver integrates from its current internal T value to a point at or beyond TOUT, then interpolates to T = TOUT and returns YB(TOUT). If ITASK is 'OneStep', then the solver takes one internal time step and returns in YB the solution at the new internal time. In this case, TOUT is used only during the first call to CVodeB to determine the direction of integration and the rough scale of the problem. In either case, the time reached by the solver is returned in T.

If quadratures were computed (see CVodeSet), CVodeB will return their values at T in the vector YQB.

On return, STATUS is one of the following:

- 0: CVodeB succeeded and no roots were found.
- 2: One of the inputs to CVodeB is illegal.
- 4: The solver took mxstep internal steps but could not reach TOUT.
The default value for mxstep is 500.
- 5: The solver could not satisfy the accuracy demanded by the user for some internal step.
- 6: Error test failures occurred too many times (MXNEF = 7) during one internal time step or occurred with $|h| = hmin$.
- 7: Convergence test failures occurred too many times (MXNCF = 10) during one internal time step or occurred with $|h| = hmin$.
- 9: The linear solver's setup routine failed in an unrecoverable manner.
- 10: The linear solver's solve routine failed in an unrecoverable manner.
- 101: Illegal attempt to call before initializing adjoint sensitivity (see CVodeMalloc).
- 104: Illegal attempt to call before CVodeMallocB.
- 108: Wrong value for TOUT.

See also CVodeSetOptions, CVodeGetstatsB

CVodeGetStats

PURPOSE

CVodeGetStats returns run statistics for the CVODES solver.

SYNOPSIS

```
function si = CVodeGetStats()
```

DESCRIPTION

CVodeGetStats returns run statistics for the CVODES solver.

Usage: STATS = CVodeGetStats

Fields in the structure STATS

- o nst - number of integration steps
- o nfe - number of right-hand side function evaluations

- o nsetups - number of linear solver setup calls
- o netf - number of error test failures
- o nni - number of nonlinear solver iterations
- o ncfn - number of convergence test failures
- o qlast - last method order used
- o qcur - current method order
- o h0used - actual initial step size used
- o hlast - last step size used
- o hcur - current step size
- o tcur - current time reached by the integrator
- o RootInfo - structure with rootfinding information
- o QuadInfo - structure with quadrature integration statistics
- o LSInfo - structure with linear solver statistics
- o FSAInfo - structure with forward sensitivity solver statistics

If rootfinding was requested, the structure RootInfo has the following fields

- o nge - number of calls to the rootfinding function
- o roots - array of integers (a value of 1 in the i-th component means that the i-th rootfinding function has a root (upon a return with status=2 from CVode)).

If quadratures were present, the structure QuadInfo has the following fields

- o nfQe - number of quadrature integrand function evaluations
- o netfQ - number of error test failures for quadrature variables

The structure LSinfo has different fields, depending on the linear solver used.

Fields in LSinfo for the 'Dense' linear solver

- o name - 'Dense'
- o njeD - number of Jacobian evaluations
- o nfeD - number of right-hand side function evaluations for difference-quotient Jacobian approximation

Fields in LSinfo for the 'Diag' linear solver

- o name - 'Diag'
- o nfeDI - number of right-hand side function evaluations for difference-quotient Jacobian approximation

Fields in LSinfo for the 'Band' linear solver

- o name - 'Band'
- o njeB - number of Jacobian evaluations
- o nfeB - number of right-hand side function evaluations for difference-quotient Jacobian approximation

Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers

- o name - 'GMRES' or 'BiCGStab'
- o nli - number of linear solver iterations
- o npe - number of preconditioner setups

- o nps - number of preconditioner solve function calls
- o ncfl - number of linear system convergence test failures
- o njeSG - number of Jacobian-vector product evaluations
- o nfeSG - number of right-hand side function evaluations for difference-quotient Jacobian-vector product approximation

If forward sensitivities were computed, the structure FSAInfo has the following fields

- o nfSe - number of sensitivity right-hand side evaluations
 - o nfeS - number of right-hand side evaluations for difference-quotient sensitivity right-hand side approximation
 - o nsetupss - number of linear solver setups triggered by sensitivity variables
 - o netfS - number of error test failures for sensitivity variables
 - o nniS - number of nonlinear solver iterations for sensitivity variables
 - o ncfns - number of convergence test failures due to sensitivity variables
-

CVodeGetStatsB

PURPOSE

`CVodeGetStatsB` returns run statistics for the backward CVODES solver.

SYNOPSIS

```
function si = CVodeGetStatsB()
```

DESCRIPTION

`CVodeGetStatsB` returns run statistics for the backward CVODES solver.

Usage: `STATS = CVodeGetStatsB`

Fields in the structure STATS

- o nst - number of integration steps
- o nfe - number of right-hand side function evaluations
- o nsetups - number of linear solver setup calls
- o netf - number of error test failures
- o nni - number of nonlinear solver iterations
- o ncfn - number of convergence test failures
- o qlast - last method order used
- o qcur - current method order
- o h0used - actual initial step size used
- o hlast - last step size used
- o hcur - current step size
- o tcur - current time reached by the integrator
- o QuadInfo - structure with quadrature integration statistics
- o LSInfo - structure with linear solver statistics

The structure LSinfo has different fields, depending on the linear solver used.

If quadratures were present, the structure QuadInfo has the following fields

- o nfQe - number of quadrature integrand function evaluations
- o netfQ - number of error test failures for quadrature variables

Fields in LSinfo for the 'Dense' linear solver

- o name - 'Dense'
- o njeD - number of Jacobian evaluations
- o nfeD - number of right-hand side function evaluations for difference-quotient Jacobian approximation

Fields in LSinfo for the 'Diag' linear solver

- o name - 'Diag'
- o nfeDI - number of right-hand side function evaluations for difference-quotient Jacobian approximation

Fields in LSinfo for the 'Band' linear solver

- o name - 'Band'
- o njeB - number of Jacobian evaluations
- o nfeB - number of right-hand side function evaluations for difference-quotient Jacobian approximation

Fields in LSinfo for the 'GMRES' and 'BiCGStab' linear solvers

- o name - 'GMRES' or 'BiCGStab'
- o nli - number of linear solver iterations
- o npe - number of preconditioner setups
- o nps - number of preconditioner solve function calls
- o ncfl - number of linear system convergence test failures
- o njeSG - number of Jacobian-vector product evaluations
- o nfeSG - number of right-hand side function evaluations for difference-quotient Jacobian-vector product approximation

CVodeGet

PURPOSE

CVodeGet extracts data from the CVODES solver memory.

SYNOPSIS

```
function varargout = CVodeGet(key, varargin)
```

DESCRIPTION

CVodeGet extracts data from the CVODES solver memory.

Usage: RET = CVodeGet (KEY [, P1 [, P2] ...])

CVodeGet returns internal CVODES information based on KEY. For some values of KEY, additional arguments may be required and/or more than one output is returned.

KEY is a string and should be one of:

- o DerivSolution - Returns a vector containing the K-th order derivative of the solution at time T. The time T and order K must be passed through the input arguments P1 and P2, respectively:

$$DKY = \text{CVodeGet}('DerivSolution', T, K)$$
- o ErrorWeights - Returns a vector containing the current error weights.

$$EWT = \text{CVodeGet}('ErrorWeights')$$
- o CheckPointsInfo - Returns an array of structures with check point information.

$$CK = \text{CVodeGet}('CheckPointInfo')$$
- o CurrentCheckPoint - Returns the address of the active check point

$$ADDR = \text{CVodeGet}('CurrentCheckPoint');$$
- o DataPointInfo - Returns information stored for interpolation at the I-th data point in between the current check points. The index I must be passed through the argument P1.
If the interpolation type was Hermite (see CVodeSetOptions), it returns two vectors, Y and YD:

$$[Y, YD] = \text{CVodeGet}('DataPointInfo', I)$$

CVodeFree

PURPOSE

CVodeFree deallocates memory for the CVODES solver.

SYNOPSIS

```
function [] = CVodeFree()
```

DESCRIPTION

CVodeFree deallocates memory for the CVODES solver.

Usage: CVodeFree

CVodeMonitor

PURPOSE

CVodeMonitor is the default CVODES monitoring function.

SYNOPSIS

```
function [new_data] = CVodeMonitor(call, T, Y, YQ, YS, data)
```

DESCRIPTION

CVodeMonitor is the default CVODES monitoring function.

To use it, set the Monitor property in CVodeSetOptions to 'CVodeMonitor' or to @CVodeMonitor and 'MonitorData' to mondata (defined as a structure).

With default settings, this function plots the evolution of the step size, method order, and various counters.

Various properties can be changed from their default values by passing to CVodeSetOptions, through the property 'MonitorData', a structure MONDATA with any of the following fields. If a field is not defined, the corresponding default value is used.

Fields in MONDATA structure:

- o stats [true | false]
 - If true, report the evolution of the step size and method order.
- o cntr [true | false]
 - If true, report the evolution of the following counters:
nst, nfe, nni, netf, ncfn (see CVodeGetStats)
- o mode ['graphical' | 'text' | 'both']
 - In graphical mode, plot the evolutions of the above quantities.
 - In text mode, print a table.
- o xaxis ['linear' | 'log']
 - Type of the time axis for the stepsize, order, and counter plots (graphical mode only).
- o sol [true | false]
 - If true, plot solution components.
- o sensi [true | false]
 - If true and if FSA is enabled, plot sensitivity components.
- o select [array of integers]
 - To plot only particular solution components, specify their indeces in the field select. If not defined, but sol=true, all components are plotted.
- o updt [integer | 50]
 - Update frequency. Data is posted in blocks of dimension n.
- o skip [integer | 0]
 - Number of integrations steps to skip in collecting data to post.
- o dir [1 | -1]
 - Specifies forward or backward integration.
- o post [true | false]
 - If false, disable all posting. This option is necessary to disable monitoring on some processors when running in parallel.

See also CVodeSetOptions, CVMonitorFn

NOTES:

1. The argument mondata is REQUIRED. Even if only the default options are desired, set mondata=struct; and pass it to CVodeSetOptions.
2. The yQ argument is currently ignored.

SOURCE CODE

```

1  function [ new_data] = CVodeMonitor( call , T, Y, YQ, YS, data)
50
51 % Radu Serban <radu@llnl.gov>
52 % Copyright (c) 2005, The Regents of the University of California.
53 % $Revision: 1.3 $Date: 2006/03/15 19:31:25 $
54
55
56 new_data = [];
57
58 if call == 0
59
60 % Initialize unspecified fields to default values.

```

```

61 data = initialize_data(data);
62
63 % Open figure windows
64 if data.post
65
66 if data.grph
67   if data.stats | data.cntr
68     data.hfg = figure;
69   end
70 % Number of subplots in figure hfg
71 if data.stats
72   data.npg = data.npg + 2;
73 end
74 if data.cntr
75   data.npg = data.npg + 1;
76 end
77 end
78
79 if data.text
80   if data.cntr | data.stats
81     data.hft = figure;
82   end
83 end
84
85 if data.sol | data.sensi
86   data.hfs = figure;
87 end
88
89 end
90
91 % Initialize other private data
92 data.i = 0;
93 data.n = 1;
94 data.t = zeros(1,data.updt);
95 if data.stats
96   data.h = zeros(1,data.updt);
97   data.q = zeros(1,data.updt);
98 end
99 if data.cntr
100   data.nst = zeros(1,data.updt);
101   data.nfe = zeros(1,data.updt);
102   data.nni = zeros(1,data.updt);
103   data.netf = zeros(1,data.updt);
104   data.ncfn = zeros(1,data.updt);
105 end
106
107 data.first = true;           % the next one will be the first call = 1
108 data.initialized = false; % the graphical windows were not initialized
109
110 new_data = data;
111
112 return;
113
114 else

```

```

115 % If this is the first call ~= 0,
116 % use Y and YS for additional initializations
117
118 if data.first
119
120 if isempty(YS)
121     data.sensi = false;
122 end
123
124 if data.sol | data.sensi
125
126 if isempty(data.select)
127
128     data.N = length(Y);
129     data.select = [1:data.N];
130
131 else
132
133     data.N = length(data.select);
134
135 end
136
137 if data.sol
138     data.y = zeros(data.N,data.updt);
139     data.nps = data.nps + 1;
140 end
141
142 if data.sensi
143     data.Ns = size(YS,2);
144     data.ys = zeros(data.N, data.Ns, data.updt);
145     data.nps = data.nps + data.Ns;
146 end
147
148 end
149
150 data.first = false;
151
152
153 end
154
155 % Extract variables from data
156
157 hfg = data.hfg;
158 hft = data.hft;
159 hfs = data.hfs;
160 npg = data.npg;
161 nps = data.nps;
162 i = data.i;
163 n = data.n;
164 t = data.t;
165 N = data.N;
166 Ns = data.Ns;
167 y = data.y;
168 ys = data.ys;

```

```

169 h      = data.h;
170 q      = data.q;
171 nst   = data.nst;
172 nfe   = data.nfe;
173 nni   = data.nni;
174 netf  = data.netf;
175 ncfn  = data.ncfn;
176
177 end
178
179
180 % Load current statistics?
181
182 if call == 1
183
184 if i ~= 0
185     i = i-1;
186     data.i = i;
187     new_data = data;
188     return;
189 end
190
191 if data.dir == 1
192     si = CVodeGetStats;
193 else
194     si = CVodeGetStatsB;
195 end
196
197 t(n) = si.tcur;
198
199 if data.stats
200     h(n) = si.hlast;
201     q(n) = si.qlast;
202 end
203
204 if data.cntr
205     nst(n) = si.nst;
206     nfe(n) = si.nfe;
207     nni(n) = si.nni;
208     netf(n) = si.netf;
209     ncfn(n) = si.ncfn;
210 end
211
212 if data.sol
213     for j = 1:N
214         y(j,n) = Y(data.select(j));
215     end
216 end
217
218 if data.sensi
219     for k = 1:Ns
220         for j = 1:N
221             ys(j,k,n) = YS(data.select(j),k);
222         end

```

```

223     end
224   end
225
226 end
227
228 % Is it time to post?
229
230 if data.post & (n == data.updt | call==2)
231
232   if call == 2
233     n = n-1;
234   end
235
236   if ~data.initialized
237
238     if (data.stats | data.cntr) & data.grph
239       graphical_init(n, hfg, npg, data.stats, data.cntr, data.dir, ...
240                     t, h, q, nst, nfe, nni, netf, ncfn, data.xaxis);
241     end
242
243     if (data.stats | data.cntr) & data.text
244       text_init(n, hft, data.stats, data.cntr, ...
245                  t, h, q, nst, nfe, nni, netf, ncfn);
246     end
247
248     if data.sol | data.sensi
249       sol_init(n, hfs, nps, data.sol, data.sensi, data.dir, data.xaxis, ...
250                  N, Ns, t, y, ys);
251     end
252
253     data.initialized = true;
254
255   else
256
257     if (data.stats | data.cntr) & data.grph
258       graphical_update(n, hfg, npg, data.stats, data.cntr, ...
259                      t, h, q, nst, nfe, nni, netf, ncfn);
260     end
261
262     if (data.stats | data.cntr) & data.text
263       text_update(n, hft, data.stats, data.cntr, ...
264                  t, h, q, nst, nfe, nni, netf, ncfn);
265     end
266
267     if data.sol
268       sol_update(n, hfs, nps, data.sol, data.sensi, N, Ns, t, y, ys);
269     end
270
271   end
272
273   if call == 2
274
275     if (data.stats | data.cntr) & data.grph
276       graphical_final(hfg, npg, data.cntr, data.stats);

```

```

277     end
278
279     if data.sol | data.sensi
280         sol_final(hfs, nps, data.sol, data.sensi, N, Ns);
281     end
282
283     return;
284
285 end
286
287 n = 1;
288
289 else
290
291     n = n + 1;
292
293 end
294
295
296 % Save updated values in data
297
298 data.i      = data.skip;
299 data.n      = n;
300 data.npg    = npg;
301 data.t      = t;
302 data.y      = y;
303 data.ys     = ys;
304 data.h      = h;
305 data.q      = q;
306 data.nst    = nst;
307 data.nfe    = nfe;
308 data.nni    = nni;
309 data.netf   = netf;
310 data.ncfn   = ncfn;
311
312 new_data = data;
313
314 return;
315
316 %——————
317
318 function data = initialize_data(data)
319
320 if ~isfield(data, 'mode')
321     data.mode = 'graphical';
322 end
323 if ~isfield(data, 'updt')
324     data.updt = 50;
325 end
326 if ~isfield(data, 'skip')
327     data.skip = 0;
328 end
329 if ~isfield(data, 'stats')
330     data.stats = true;

```

```

331 end
332 if ~isfield(data, 'cntr')
333     data.cntr = true;
334 end
335 if ~isfield(data, 'sol')
336     data.sol = false;
337 end
338 if ~isfield(data, 'sensi')
339     data.sensi = false;
340 end
341 if ~isfield(data, 'select')
342     data.select = [];
343 end
344 if ~isfield(data, 'xaxis')
345     data.xaxis = 'log';
346 end
347 if ~isfield(data, 'dir')
348     data.dir = 1;
349 end
350 if ~isfield(data, 'post')
351     data.post = true;
352 end
353
354 data.grph = true;
355 data.text = true;
356 if strcmp(data.mode, 'graphical')
357     data.text = false;
358 end
359 if strcmp(data.mode, 'text')
360     data.grph = false;
361 end
362
363 if ~data.sol & ~data.sensi
364     data.select = [];
365 end
366
367 % Other initializations
368 data.npg = 0;
369 data.nps = 0;
370 data.hfg = 0;
371 data.hft = 0;
372 data.hfs = 0;
373 data.h = 0;
374 data.q = 0;
375 data.nst = 0;
376 data.nfe = 0;
377 data.nni = 0;
378 data.netf = 0;
379 data.ncfn = 0;
380 data.N = 0;
381 data.Ns = 0;
382 data.y = 0;
383 data.ys = 0;
384

```

```

385 %_
386
387 function [] = graphical_init(n, hfg, npg, stats, cntr, dir, ...
388                             t, h, q, nst, nfe, nni, netf, ncfn, xaxis)
389
390 fig_name = 'CVODES-run_statistics';
391
392 % If this is a parallel job, look for the MPI rank in the global
393 % workspace and append it to the figure name
394
395 global sundials_MPI_rank
396
397 if ~isempty(sundials_MPI_rank)
398     fig_name = sprintf('%s_(PE%d)',fig_name,sundials_MPI_rank);
399 end
400
401 figure(hfg);
402 set(hfg,'Name',fig_name);
403 set(hfg,'color',[1 1 1]);
404 pl = 0;
405
406 % Time label and figure title
407
408 if dir==1
409     tlab = '\rightarrow\leftarrow\rightarrow';
410 else
411     tlab = '\leftarrow\rightarrow\leftarrow';
412 end
413
414 % Step size and order
415 if stats
416     pl = pl+1;
417     subplot(npg,1,pl)
418     semilogy(t(1:n),abs(h(1:n)),'-');
419     if strcmp(xaxis,'log')
420         set(gca,'XScale','log');
421     end
422     hold on;
423     box on;
424     grid on;
425     xlabel(tlab);
426     ylabel('|Step-size|');
427
428     pl = pl+1;
429     subplot(npg,1,pl)
430     plot(t(1:n),q(1:n),'-');
431     if strcmp(xaxis,'log')
432         set(gca,'XScale','log');
433     end
434     hold on;
435     box on;
436     grid on;
437     xlabel(tlab);
438     ylabel('Order');

```

```

439 end
440
441 % Counters
442 if cntr
443 pl = pl+1;
444 subplot(npg,1,pl)
445 plot(t(1:n),nst(1:n),'k-');
446 hold on;
447 plot(t(1:n),nfe(1:n),'b-');
448 plot(t(1:n),nni(1:n),'r-');
449 plot(t(1:n),netf(1:n),'g-');
450 plot(t(1:n),ncfn(1:n),'c-');
451 if strcmp(xaxis,'log')
452 set(gca,'XScale','log');
453 end
454 box on;
455 grid on;
456 xlabel(tlab);
457 ylabel('Counters');
458 end
459
460 drawnow;
461
462 %
463
464 function [] = graphical_update(n, hfg, npg, stats, cntr, ...
465 t, h, q, nst, nfe, nni, netf, ncfn)
466
467 figure(hfg);
468 pl = 0;
469
470 % Step size and order
471 if stats
472 pl = pl+1;
473 subplot(npg,1,pl)
474 hc = get(gca,'Children');
475 xd = [get(hc,'XData') t(1:n)];
476 yd = [get(hc,'YData') abs(h(1:n))];
477 set(hc,'XData',xd,'YData',yd);
478
479 pl = pl+1;
480 subplot(npg,1,pl)
481 hc = get(gca,'Children');
482 xd = [get(hc,'XData') t(1:n)];
483 yd = [get(hc,'YData') q(1:n)];
484 set(hc,'XData',xd,'YData',yd);
485 end
486
487 % Counters
488 if cntr
489 pl = pl+1;
490 subplot(npg,1,pl)
491 hc = get(gca,'Children');
492 % Attention: Children are loaded in reverse order!

```

```

493 xd = [get(hc(1), 'XData') t(1:n)];
494 yd = [get(hc(1), 'YData') ncfn(1:n)];
495 set(hc(1), 'XData', xd, 'YData', yd);
496 xd = [get(hc(2), 'XData') netf(1:n)];
497 set(hc(2), 'XData', xd, 'YData', yd);
498 yd = [get(hc(3), 'YData') nni(1:n)];
499 set(hc(3), 'XData', xd, 'YData', yd);
500 yd = [get(hc(4), 'YData') nfe(1:n)];
501 set(hc(4), 'XData', xd, 'YData', yd);
502 yd = [get(hc(5), 'YData') nst(1:n)];
503 set(hc(5), 'XData', xd, 'YData', yd);
504 end
505
506 drawnow;
507
508 %
509
510 function [] = graphical_final(hfg, npg, stats, cntr)
511 figure(hfg);
512 pl = 0;
513
514 if stats
515 pl = pl+1;
516 subplot(npg, 1, pl)
517 hc = get(gca, 'Children');
518 xd = get(hc, 'XData');
519 set(gca, 'XLim', sort([xd(1) xd(end)]));
520
521 pl = pl+1;
522 subplot(npg, 1, pl)
523 ylim = get(gca, 'YLim');
524 ylim(1) = ylim(1) - 1;
525 ylim(2) = ylim(2) + 1;
526 set(gca, 'YLim', ylim);
527 set(gca, 'XLim', sort([xd(1) xd(end)]));
528
529 end
530
531 if cntr
532 pl = pl+1;
533 subplot(npg, 1, pl)
534 hc = get(gca, 'Children');
535 xd = get(hc(1), 'XData');
536 set(gca, 'XLim', sort([xd(1) xd(end)]));
537 legend('nst', 'nfe', 'nni', 'netf', 'ncfn', 2);
538
539 end
540 %
541
542 function [] = text_init(n, hft, stats, cntr, t, h, q, nst, nfe, nni, netf, ncfn)
543 fig_name = 'CVODES-run-statistics';
544 % If this is a parallel job, look for the MPI rank in the global

```

```

547 % workspace and append it to the figure name
548
549 global sundials_MPI_rank
550
551 if ~isempty(sundials_MPI_rank)
552 fig_name = sprintf('%s_(PE%d)',fig_name,sundials_MPI_rank);
553 end
554
555 figure(hft);
556 set(hft,'Name',fig_name);
557 set(hft,'color',[1 1 1]);
558 set(hft,'MenuBar','none');
559 set(hft,'Resize','off');
560
561 % Create text box
562
563 margins=[10 10 50 50]; % left , right , top , bottom
564 pos=get(hft,'position');
565 tbpos=[margins(1) margins(4) pos(3)-margins(1)-margins(2) ...
566 pos(4)-margins(3)-margins(4)];
567 tbpos(tbpos<1)=1;
568
569 htb=uicontrol(hft,'style','listbox','position',tbpos,'tag','textbox');
570 set(htb,'BackgroundColor',[1 1 1]);
571 set(htb,'SelectionHighlight','off');
572 set(htb,'FontName','courier');
573
574 % Create table head
575
576 tpos = [tbpos(1) tbpos(2)+tbpos(4)+10 tbpos(3) 20];
577 ht=uicontrol(hft,'style','text','position',tpos,'tag','text');
578 set(ht,'BackgroundColor',[1 1 1]);
579 set(ht,'HorizontalAlignment','left');
580 set(ht,'FontName','courier');
581 newline = 'time step order | nfe nni netfn';
582 set(ht,'String',newline);
583
584 % Create OK button
585
586 bsize=[60,28];
587 badjustpos=[0,25];
588 bpos=[pos(3)/2-bsize(1)/2+badjustpos(1)-bsize(2)/2+badjustpos(2)...
589 bsize(1) bsize(2)];
590 bpos=round(bpos);
591 bpos(bpos<1)=1;
592 hb=uicontrol(hft,'style','pushbutton','position',bpos,...
593 'string','Close','tag','okaybutton');
594 set(hb,'callback','close');
595
596 % Save handles
597
598 handles=guihandles(hft);
599 guidata(hft,handles);
600

```

```

601 for i = 1:n
602     newline = ' ';
603     if stats
604         newline = sprintf('%.10.3e %.10.3e %.1d | ', t(i), h(i), q(i));
605     end
606     if cntr
607         newline = sprintf('%s %.5d %.5d %.5d %.5d %.5d' ,...
608                         newline, nst(i), nfe(i), nni(i), netf(i), ncfn(i));
609     end
610     string = get(handles.textbox, 'String');
611     string{end+1}=newline;
612     set(handles.textbox, 'String', string);
613 end

614 drawnow
615 %
616 %
617 function [] = text_update(n, hft, stats, cntr, t, h, q, nst, nfe, nni, netf, ncfn)
618
619 figure(hft);
620
621 handles=guidata(hft);
622
623 for i = 1:n
624     if stats
625         newline = sprintf('%.10.3e %.10.3e %.1d | ', t(i), h(i), q(i));
626     end
627     if cntr
628         newline = sprintf('%s %.5d %.5d %.5d %.5d %.5d' ,...
629                         newline, nst(i), nfe(i), nni(i), netf(i), ncfn(i));
630     end
631     string = get(handles.textbox, 'String');
632     string{end+1}=newline;
633     set(handles.textbox, 'String', string);
634 end
635
636 drawnow
637 %
638 %
639 function [] = sol_init(n, hfs, nps, sol, sensi, dir, xaxis, N, Ns, t, y, ys)
640
641 fig_name = 'CVODES_solution';
642
643 % If this is a parallel job, look for the MPI rank in the global
644 % workspace and append it to the figure name
645
646 global sundials_MPI_rank
647
648 if ~isempty(sundials_MPI_rank)
649     fig_name = sprintf('%s_(PE_%d)', fig_name, sundials_MPI_rank);
650 end
651
652
653
654
```

```

655
656 figure(hfs);
657 set(hfs, 'Name', fig_name);
658 set(hfs, 'color', [1 1 1]);
659
660 % Time label
661
662 if dir==1
663 tlab = '\rightarrowleftarrow\rightarrow';
664 else
665 tlab = '\leftarrow\rightarrowleftarrow\leftarrow';
666 end
667
668 % Get number of colors in colormap
669 map = colormap;
670 ncols = size(map,1);
671
672 % Initialize current subplot counter
673 pl = 0;
674
675 if sol
676
677 pl = pl+1;
678 subplot(nps,1,pl);
679 hold on;
680
681 for i = 1:N
682 hp = plot(t(1:n),y(i,1:n),'-');
683 ic = 1+(i-1)*floor(ncols/N);
684 set(hp, 'Color', map(ic,:));
685 end
686 if strcmp(xaxis, 'log')
687 set(gca, 'XScale', 'log');
688 end
689 box on;
690 grid on;
691 xlabel(tlab);
692 ylabel('y');
693 title('Solution');
694 end
695
696 if sensi
697
698 for is = 1:Ns
699
700 pl = pl+1;
701 subplot(nps,1,pl);
702 hold on;
703
704 ys_crt = ys(:,is,1:n);
705 for i = 1:N
706 hp = plot(t(1:n),ys_crt(i,1:n),'-');
707 ic = 1+(i-1)*floor(ncols/N);

```

```

709     set(hp, 'Color', map(ic,:));
710 end
711 if strcmp(xaxis, 'log')
712     set(gca, 'XScale', 'log');
713 end
714 box on;
715 grid on;
716 xlabel(tlab);
717 str = sprintf('s-%d',is); ylabel(str);
718 str = sprintf('Sensitivity %d',is); title(str);
719
720 end
721
722 end
723
724
725 drawnow;
726
727 %
728
729 function [] = sol_update(n, hfs, nps, sol, sensi, N, Ns, t, y, ys)
730
731 figure(hfs);
732
733 pl = 0;
734
735 if sol
736
737     pl = pl+1;
738     subplot(nps,1,pl);
739
740     hc = get(gca, 'Children');
741     xd = [get(hc(1), 'XData') t(1:n)];
742 % Attention: Children are loaded in reverse order!
743     for i = 1:N
744         yd = [get(hc(i), 'YData') y(N-i+1,1:n)];
745         set(hc(i), 'XData', xd, 'YData', yd);
746     end
747
748 end
749
750 if sensi
751
752     for is = 1:Ns
753
754         pl = pl+1;
755         subplot(nps,1,pl);
756
757         ys_crt = ys(:,is,:);
758
759         hc = get(gca, 'Children');
760         xd = [get(hc(1), 'XData') t(1:n)];
761 % Attention: Children are loaded in reverse order!
762         for i = 1:N

```

```

763     yd = [ get(hc(i), 'YData') ys_crt(N-i+1,1:n)];  

764     set(hc(i), 'XData', xd, 'YData', yd);  

765 end  

766  

767 end  

768  

769  

770  

771 drawnow;  

772  

773  

774  

775 %  

776  

777 function [] = sol_final(hfs, nps, sol, sensi, N, Ns)  

778 figure(hfs);  

779  

780 pl = 0;  

781  

782 if sol  

783  

784 pl = pl +1;  

785 subplot(nps,1,pl);  

786  

787 hc = get(gca, 'Children');  

788 xd = get(hc(1), 'XData');  

789 set(gca, 'XLim', sort([xd(1) xd(end)]));  

790  

791 ylim = get(gca, 'YLim');  

792 addon = 0.1*abs(ylim(2)-ylim(1));  

793 ylim(1) = ylim(1) + sign(ylim(1))*addon;  

794 ylim(2) = ylim(2) + sign(ylim(2))*addon;  

795 set(gca, 'YLim', ylim);  

796  

797 for i = 1:N  

798     cstring{i} = sprintf('y-%d', i);  

799 end  

800 legend(cstring);  

801  

802 end  

803  

804 if sensi  

805  

806 for is = 1:Ns  

807  

808 pl = pl+1;  

809 subplot(nps,1,pl);  

810  

811 hc = get(gca, 'Children');  

812 xd = get(hc(1), 'XData');  

813 set(gca, 'XLim', sort([xd(1) xd(end)]));  

814  

815 ylim = get(gca, 'YLim');  

816

```

```

817     addon = 0.1*abs( ylim(2)-ylim(1));
818     ylim(1) = ylim(1) + sign(ylim(1))*addon;
819     ylim(2) = ylim(2) + sign(ylim(2))*addon;
820     set(gca, 'YLim', ylim);
821
822     for i = 1:N
823         cstring{i} = sprintf('s%d-%d', is, i);
824     end
825     legend(cstring);
826
827 end
828
829 end
830
831 drawnow

```

2.2 Function types

CVBandJacFn

PURPOSE

CVBandJacFn - type for user provided banded Jacobian function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVBandJacFn - type for user provided banded Jacobian function.

IVP Problem

The function BJACFUN must be defined as

FUNCTION [J, FLAG] = BJACFUN(T, Y, FY)

and must return a matrix J corresponding to the banded Jacobian of $f(t,y)$.

The input argument FY contains the current value of $f(t,y)$.

If a user data structure DATA was specified in CVodeMalloc, then

BJACFUN must be defined as

FUNCTION [J, FLAG, NEW_DATA] = BJACFUN(T, Y, FY, DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the matrix J, the BJACFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function BJACFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

Adjoint Problem

The function BJACFUNB must be defined either as

FUNCTION [JB, FLAG] = BJACFUNB(T, Y, YB, FYB)

or as

FUNCTION [JB, FLAG, NEW_DATA] = BJACFUNB(T, Y, YB, FYB, DATA)

depending on whether a user data structure DATA was specified in CVodeMalloc. In either case, it must return the matrix JB, the Jacobian of $f_B(t,y,yB)$, with respect to yB. The input argument FYB contains the current value of $f(t,y,yB)$.

The function BJACFUNB must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also CVodeSetOptions

See the CVODES user guide for more information on the structure of

a banded Jacobian.

NOTE: BJACFUN and BJACFUNB are specified through the property JacobianFn to CVodeSetOptions and are used only if the property LinearSolver was set to 'Band'.

CVDenseJacFn

PURPOSE

CVDenseJacFn – type for user provided dense Jacobian function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVDenseJacFn – type for user provided dense Jacobian function.

IVP Problem

The function DJACFUN must be defined as

FUNCTION [J, FLAG] = DJACFUN(T, Y, FY)

and must return a matrix J corresponding to the Jacobian of $f(t,y)$.

The input argument FY contains the current value of $f(t,y)$.

If a user data structure DATA was specified in CVodeMalloc, then DJACFUN must be defined as

FUNCTION [J, FLAG, NEW_DATA] = DJACFUN(T, Y, FY, DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the matrix J, the DJACFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function DJACFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

Adjoint Problem

The function DJACFUNB must be defined either as

FUNCTION [JB, FLAG] = DJACFUNB(T, Y, YB, FYB)

or as

FUNCTION [JB, FLAG, NEW_DATA] = DJACFUNB(T, Y, YB, FYB, DATA)

depending on whether a user data structure DATA was specified in CVodeMalloc. In either case, it must return the matrix JB, the Jacobian of $f_B(t,y,y_B)$, with respect to y_B . The input argument FYB contains the current value of $f(t,y,y_B)$.

The function DJACFUNB must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also CVodeSetOptions

NOTE: DJACFUN and DJACFUNB are specified through the property JacobianFn to CVodeSetOptions and are used only if the property LinearSolver was set to 'Dense'.

CVGcommFn

PURPOSE

CVGcommFn - type for user provided communication function (BBDPre).

SYNOPSIS

This is a script file.

DESCRIPTION

CVGcommFn - type for user provided communication function (BBDPre).

IVP Problem

The function GCOMFUN must be defined as

FUNCTION FLAG = GCOMFUN(T, Y)

and can be used to perform all interprocess communication necessary to evaluate the approximate right-hand side function for the BBDPre preconditioner module.

If a user data structure DATA was specified in CVodeMalloc, then GCOMFUN must be defined as

FUNCTION [FLAG, NEW_DATA] = GCOMFUN(T, Y, DATA)

If the local modifications to the user data structure are needed in other user-provided functions then the GCOMFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function GCOMFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

Adjoint Problem

The function GCOMFUNB must be defined either as

FUNCTION FLAG = GCOMFUNB(T, Y, YB)

or as

FUNCTION [FLAG, NEW_DATA] = GCOMFUNB(T, Y, YB, DATA)

depending on whether a user data structure DATA was specified in CVodeMalloc.

The function GCOMFUNB must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also CVGlocalFn, CVodeSetOptions

NOTES:

GCOMFUN and GCOMFUNB are specified through the GcommFn property in CVodeSetOptions and are used only if the property PrecModule is set to 'BBDPre'.

Each call to GCOMFUN is preceded by a call to the RHS function ODEFUN with the same arguments T and Y (and YB in the case of GCOMFUNB). Thus GCOMFUN can omit any communication done by ODEFUN if relevant to the evaluation of G by GLOCFUN. If all necessary communication was done by ODEFUN, GCOMFUN need not be provided.

CVGlocalFn

PURPOSE

CVGlocalFn - type for user provided RHS approximation function (BBDPre).

SYNOPSIS

This is a script file.

DESCRIPTION

CVGlocalFn - type for user provided RHS approximation function (BBDPre).

IVP Problem

The function GLOCFUN must be defined as

FUNCTION [GLOC, FLAG] = GLOCFUN(T,Y)

and must return a vector GLOC corresponding to an approximation to $f(t,y)$ which will be used in the BBDPRE preconditioner module. The case where G is mathematically identical to F is allowed.

If a user data structure DATA was specified in CVodeMalloc, then GLOCFUN must be defined as

FUNCTION [GLOC, FLAG, NEW_DATA] = GLOCFUN(T,Y,DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector G, the GLOCFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function GLOCFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

Adjoint Problem

The function GLOCFUNB must be defined either as

FUNCTION [GLOCB, FLAG] = GLOCFUNB(T,Y,YB)

or as

FUNCTION [GLOCB, FLAG, NEW_DATA] = GLOCFUNB(T,Y,YB,DATA)

depending on whether a user data structure DATA was specified in CVodeMalloc. In either case, it must return the vector GLOCB corresponding to an approximation to $f_B(t,y,yB)$.

The function GLOCFUNB must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also CVGcommFn, CVodeSetOptions

NOTE: GLOCFUN and GLOCFUNB are specified through the GlocalFn property in CVodeSetOptions and are used only if the property PrecModule is set to 'BBDPre'.

CVMonitorFn

PURPOSE

CVMonitorFn - type for user provided monitoring function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVMonitorFn - type for user provided monitoring function.

The function MONFUN must be defined as

FUNCTION [] = MONFUN(CALL, T, Y, YQ, YS)

It is called after every internal CVode step and can be used to monitor the progress of the solver. MONFUN is called with CALL=0 from CVodeMalloc at which time it should initialize itself and it is called with CALL=2 from CVodeFree. Otherwise, CALL=1.

It receives as arguments the current time T, solution vector Y, and, if they were computed, quadrature vector YQ, and forward sensitivity matrix YS. If YQ and/or YS were not computed they are empty here.

If additional data is needed inside MONFUN, it must be defined as

FUNCTION NEW_MONDATA = MONFUN(CALL, T, Y, YQ, YS, MONDATA)

If the local modifications to the user data structure need to be saved (e.g. for future calls to MONFUN), then MONFUN must set NEW_MONDATA. Otherwise, it should set NEW_MONDATA=[] (do not set NEW_MONDATA = DATA as it would lead to unnecessary copying).

A sample monitoring function, CVodeMonitor, is provided with CVODES.

See also CVodeSetOptions, CVodeMonitor

NOTES:

MONFUN is specified through the MonitorFn property in CVodeSetOptions. If this property is not set, or if it is empty, MONFUN is not used. MONDATA is specified through the MonitorData property in CVodeSetOptions.

If MONFUN is used on the backward integration phase, YS will always be empty.

See CVodeMonitor for an example of using MONDATA to write a single monitoring function that works both for the forward and backward integration phases.

CVQuadRhsFn

PURPOSE

CVQuadRhsFn - type for user provided quadrature RHS function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVQuadRhsFn - type for user provided quadrature RHS function.

IVP Problem

The function ODEQFUN must be defined as

FUNCTION [YQD, FLAG] = ODEQFUN(T,Y)

and must return a vector YQD corresponding to $f_Q(t,y)$, the integrand for the integral to be evaluated.

If a user data structure DATA was specified in CVodeMalloc, then ODEQFUN must be defined as

FUNCTION [YQD, FLAG, NEW_DATA] = ODEQFUN(T,Y,DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector YQD, the ODEQFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function ODEQFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

Adjoint Problem

The function ODEQFUNB must be defined either as

FUNCTION [YQBD, FLAG] = ODEQFUNB(T,Y,YB)

or as

FUNCTION [YQBD, FLAG, NEW_DATA] = ODEQFUNB(T,Y,YB,DATA)

depending on whether a user data structure DATA was specified in CVodeMalloc. In either case, it must return the vector YQBD corresponding to $f_{QB}(t,y,y_B)$, the integrand for the integral to be evaluated on the backward phase.

The function ODEQFUNB must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error

occurred.

See also `CVodeSetOptions`

NOTE: `ODEQFUN` and `ODEQFUNB` are specified through the property `QuadRhsFn` to `CVodeSetOptions` and are used only if the property `Quadratures` was set to 'on'.

CVRhsFn

PURPOSE

`CVRhsFn` - type for user provided RHS type

SYNOPSIS

This is a script file.

DESCRIPTION

`CVRhsFn` - type for user provided RHS type

IVP Problem

The function `ODEFUN` must be defined as

FUNCTION [YD, FLAG] = ODEFUN(T,Y)

and must return a vector `YD` corresponding to $f(t,y)$.

If a user data structure `DATA` was specified in `CVodeMalloc`, then `ODEFUN` must be defined as

FUNCTION [YD, FLAG, NEW_DATA] = ODEFUN(T,Y,DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector `YD`, the `ODEFUN` function must also set `NEW_DATA`. Otherwise, it should set `NEW_DATA=[]` (do not set `NEW_DATA = DATA` as it would lead to unnecessary copying).

The function `ODEFUN` must set `FLAG=0` if successful, `FLAG<0` if an unrecoverable failure occurred, or `FLAG>0` if a recoverable error occurred.

Adjoint Problem

The function `ODEFUNB` must be defined either as

FUNCTION [YBD, FLAG] = ODEFUNB(T,Y,YB)

or as

FUNCTION [YBD, FLAG, NEW_DATA] = ODEFUNB(T,Y,YB,DATA)

depending on whether a user data structure `DATA` was specified in `CVodeMalloc`. In either case, it must return the vector `YBD` corresponding to $f_B(t,y,y_B)$.

The function `ODEFUNB` must set `FLAG=0` if successful, `FLAG<0` if an unrecoverable failure occurred, or `FLAG>0` if a recoverable error occurred.

See also CVodeMalloc, CVodeMallocB

NOTE: ODEFUN and ODEFUNB are specified through the CVodeMalloc and CVodeMallocB functions, respectively.

CVRootFn

PURPOSE

CVRootFn - type for user provided root-finding function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVRootFn - type for user provided root-finding function.

The function ROOTFUN must be defined as

FUNCTION [G, FLAG] = ROOTFUN(T,Y)

and must return a vector G corresponding to $g(t,y)$.

If a user data structure DATA was specified in CVodeMalloc, then ROOTFUN must be defined as

FUNCTION [G, FLAG, NEW_DATA] = ROOTFUN(T,Y,DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector G, the ROOTFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA= [] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function ROOTFUN must set FLAG=0 if successful, or FLAG^=0 if a failure occurred.

See also CVodeSetOptions

NOTE: ROOTFUN is specified through the RootsFn property in CVodeSetOptions and is used only if the property NumRoots is a positive integer.

CVSensRhsFn

PURPOSE

CVSensRhsFn - type for user provided sensitivity RHS function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVSensRhsFn - type for user provided sensitivity RHS function.

The function ODESFUN must be defined as

```
FUNCTION [YSD, FLAG] = ODESFUN(T,Y,YD,YS)
```

and must return a matrix YSD corresponding to $f_S(t,y,yS)$.

If a user data structure DATA was specified in CVodeMalloc, then ODESFUN must be defined as

```
FUNCTION [YSD, FLAG, NEW_DATA] = ODESFUN(T,Y,YD,YS,DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the matrix YSD, the ODESFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA= [] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function ODESFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also CVodeSetOptions

NOTE: ODESFUN is specified through the property FSARhsFn to CVodeSetOptions and is used only if the property SensiAnalysis was set to 'FSA' and if the property FSARhsType was set to 'All'.

CVJacTimesVecFn

PURPOSE

CVJacTimesVecFn - type for user provided Jacobian times vector function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVJacTimesVecFn - type for user provided Jacobian times vector function.

IVP Problem

The function JTVFUN must be defined as

```
FUNCTION [JV, FLAG] = JTVFUN(T,Y,FY,V)
```

and must return a vector JV corresponding to the product of the Jacobian of $f(t,y)$ with the vector v.

The input argument FY contains the current value of $f(t,y)$.

If a user data structure DATA was specified in CVodeMalloc, then JTVFUN must be defined as

```
FUNCTION [JV, FLAG, NEW_DATA] = JTVFUN(T,Y,FY,V,DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector JV, the JTVFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA= [] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function JTVFUN must set FLAG=0 if successful, or FLAG^=0 if a failure occurred.

Adjoint Problem

The function JTVFUNB must be defined either as

```
FUNCTION [JVB, FLAG] = JTVFUNB(T,Y,YB,FYB,VB)
```

or as

```
FUNCTION [JVB, FLAG, NEW_DATA] = JTVFUNB(T,Y,YB,FYB,VB,DATA)
```

depending on whether a user data structure DATA was specified in CVodeMalloc. In either case, it must return the vector JVB, the product of the Jacobian of fB(t,y,yB) with respect to yB and a vector vB. The input argument FYB contains the current value of f(t,y,yB).

The function JTVFUNB must set FLAG=0 if successful, or FLAG^=0 if a failure occurred.

See also CVodeSetOptions

NOTE: JTVFUN and JTVFUNB are specified through the property JacobianFn to CVodeSetOptions and are used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.

CVPrecSetupFn

PURPOSE

CVPrecSetupFn - type for user provided preconditioner setup function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVPrecSetupFn - type for user provided preconditioner setup function.

The user-supplied preconditioner setup function PSETFUN and the user-supplied preconditioner solve function PSOLFUN together must define left and right preconditioner matrices P1 and P2 (either of which may be trivial), such that the product P1*P2 is an approximation to the Newton matrix M = I - gamma*J. Here J is the system Jacobian J = df/dy, and gamma is a scalar proportional to the integration step size h. The solution of systems P z = r, with P = P1 or P2, is to be carried out by the PrecSolve function, and PSETFUN is to do any necessary setup operations.

The user-supplied preconditioner setup function PSETFUN is to evaluate and preprocess any Jacobian-related data needed by the preconditioner solve function PSOLFUN. This might include forming a crude approximate Jacobian, and performing an LU factorization on the resulting approximation to M. This function will not be called in

advance of every call to PSOLFUN, but instead will be called only as often as necessary to achieve convergence within the Newton iteration. If the PSOLFUN function needs no preparation, the PSETFUN function need not be provided.

For greater efficiency, the PSETFUN function may save Jacobian-related data and reuse it, rather than generating it from scratch. In this case, it should use the input flag JOK to decide whether to recompute the data, and set the output flag JCUR accordingly.

Each call to the PSETFUN function is preceded by a call to ODEFUN with the same (t,y) arguments. Thus the PSETFUN function can use any auxiliary data that is computed and saved by the ODEFUN function and made accessible to PSETFUN.

IVP Problem

The function PSETFUN must be defined as

```
FUNCTION [JCUR, FLAG] = PSETFUN(T,Y,FY,JOK,GAMMA)
```

and must return a logical flag JCUR (true if Jacobian information was recomputed and false if saved data was reused). If PSETFUN was successful, it must return FLAG=0. For a recoverable error (in which case the setup will be retried) it must set FLAG to a positive integer value. If an unrecoverable error occurs, it must set FLAG to a negative value, in which case the integration will be halted. The input argument FY contains the current value of f(t,y). If the input logical flag JOK is false, it means that Jacobian-related data must be recomputed from scratch. If it is true, it means that Jacobian data, if saved from the previous PSETFUN call can be reused (with the current value of GAMMA).

If a user data structure DATA was specified in CVodeMalloc, then PSETFUN must be defined as

```
FUNCTION [JCUR, FLAG, NEW_DATA] = PSETFUN(T,Y,FY,JOK,GAMMA,DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the flags JCUR and FLAG, the PSETFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

Adjoint Problem

The function PSETFUNB must be defined either as

```
FUNCTION [JCURB, FLAG] = PSETFUNB(T,Y,YB,FYB,JOK,GAMMAB)
```

or as

```
FUNCTION [JCURB, FLAG, NEW_DATA] = PSETFUNB(T,Y,YB,FYB,JOK,GAMMAB,DATA)
```

depending on whether a user data structure DATA was specified in CVodeMalloc. In either case, it must return the flags JCURB and FLAG.

See also CVPrecSolveFn, CVodeSetOptions

NOTE: PSETFUN and PSETFUNB are specified through the property PrecSetupFn to CVodeSetOptions and are used only if the property

LinearSolver was set to 'GMRES' or 'BiCGStab' and if the property PrecType is not 'None'.

CVPrecSolveFn

PURPOSE

CVPrecSolveFn - type for user provided preconditioner solve function.

SYNOPSIS

This is a script file.

DESCRIPTION

CVPrecSolveFn - type for user provided preconditioner solve function.

The user-supplied preconditioner solve function PSOLFN is to solve a linear system $P z = r$ in which the matrix P is one of the preconditioner matrices P_1 or P_2 , depending on the type of preconditioning chosen.

IVP Problem

The function PSOLFUN must be defined as

FUNCTION [Z, FLAG] = PSOLFUN(T,Y,FY,R)

and must return a vector Z containing the solution of $Pz=r$.

If PSOLFUN was successful, it must return FLAG=0. For a recoverable error (in which case the step will be retried) it must set FLAG to a positive value. If an unrecoverable error occurs, it must set FLAG to a negative value, in which case the integration will be halted. The input argument FY contains the current value of $f(t,y)$.

If a user data structure DATA was specified in CVodeMalloc, then PSOLFUN must be defined as

FUNCTION [Z, FLAG, NEW_DATA] = PSOLFUN(T,Y,FY,R,DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector Z and the flag FLAG, the PSOLFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

Adjoint Problem

The function PSOLFUNB must be defined either as

FUNCTION [ZB, FLAG] = PSOLFUNB(T,Y,YB,FYB,RB)

or as

FUNCTION [ZB, FLAG, NEW_DATA] = PSOLFUNB(T,Y,YB,FYB,RB,DATA)

depending on whether a user data structure DATA was specified in CVodeMalloc. In either case, it must return the vector ZB and the flag FLAG.

See also CVPrecSetupFn, CVodeSetOptions

NOTE: PSOLFUNK and PSOLFUNKB are specified through the property PrecSolveFn to CVodeSetOptions and are used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab' and if the property PrecType is not 'None'.

3 MATLAB Interface to KINSOL

The MATLAB interface to KINSOL provides access to all functionality of the KINSOL solver.

The interface consists of 5 user-callable functions. The user must provide several required and optional user-supplied functions which define the problem to be solved. The user-callable functions and the types of user-supplied functions are listed in Table 2 and fully documented later in this section. For more in depth details, consult also the KINSOL user guide [1].

To illustrate the use of the KINSOL MATLAB interface, several example problems are provided with SUNDIALSTB, both for serial and parallel computations. Most of them are MATLAB translations of example problems provided with KINSOL.

Table 2: KINSOL MATLAB interface functions

Functions	KINSetOptions KINMalloc KINSol KINGetStats KINFree	creates an options structure for KINSOL. allocates and initializes memory for KINSOL. solves the nonlinear problem. returns statistics for the KINSOL solver. deallocates memory for the KINSOL solver.
Function types	KINSysFn KINDenseJacFn KINBandJacFn KINJacTimesVecFn KINPrecSetupFn KINPrecSolveFn KINGlocalFn KINGcommFn	system function dense Jacobian function banded Jacobian function Jacobian times vector function preconditioner setup function preconditioner solve function system approximation function (BBDPre) communication function (BBDPre)

3.1 Interface functions

KINSetOptions

PURPOSE

KINSetOptions creates an options structure for KINSOL.

SYNOPSIS

```
function options = KINSetOptions(varargin)
```

DESCRIPTION

KINSetOptions creates an options structure for KINSOL.

Usage:

options = KINSetOptions('NAME1',VALUE1,'NAME2',VALUE2,...) creates a KINSOL options structure options in which the named properties have the specified values. Any unspecified properties have default values. It is sufficient to type only the leading characters that uniquely identify the property. Case is ignored for property names.

options = KINSetOptions(olddoptions,'NAME1',VALUE1,...) alters an existing options structure oldoptions.

options = KINSetOptions(olddoptions,newoptions) combines an existing options structure oldoptions with a new options structure newoptions. Any new properties overwrite corresponding old properties.

KINSetOptions with no input arguments displays all property names and their possible values.

KINSetOptions properties

(See also the KINSOL User Guide)

Verbose - verbose output [false | true]

Specifies whether or not KINSOL should output additional information

MaxNumIter - maximum number of nonlinear iterations [scalar | 200]

Specifies the maximum number of iterations that the nonlinear solver is allowed to take.

FuncRelErr - relative residual error [scalar | eps]

Specifies the realative error in computing $f(y)$ when used in difference quotient approximation of matrix-vector product $J(y)*v$.

FuncNormTol - residual stopping criteria [scalar | $\text{eps}^{(1/3)}$]

Specifies the stopping tolerance on $\|f\text{scale} \cdot \text{ABS}(f(y))\|_{\text{L-infinity}}$

ScaledStepTol - step size stopping criteria [scalar | $\text{eps}^{(2/3)}$]

Specifies the stopping tolerance on the maximum scaled step length:

$$\frac{\|y_{(k+1)} - y_k\|}{\|y_{(k+1)}\| + \text{yscale}} \|_{\text{L-infinity}}$$

MaxNewtonStep - maximum Newton step size [scalar | 0.0]

Specifies the maximum allowable value of the scaled length of the Newton step.
InitialSetup - initial call to linear solver setup [false | true]
 Specifies whether or not KINSol makes an initial call to the linear solver setup function.
MaxNumSetups - [scalar | 10]
 Specifies the maximum number of nonlinear iterations between calls to the linear solver setup function (i.e. Jacobian/preconditioner evaluation)
MaxNumSubSetups - [scalar | 5]
 Specifies the maximum number of nonlinear iterations between checks by the nonlinear residual monitoring algorithm (specifies length of subintervals).
 NOTE: MaxNumSetups should be a multiple of MaxNumSubSetups.
MaxNumBetaFails - maximum number of beta-condition failures [scalar | 10]
 Specifies the maximum number of beta-condition failures in the line search algorithm.
EtaForm - Inexact Newton method [Constant | Type2 | Type1]
 Specifies the method for computing the eta coefficient used in the calculation of the linear solver convergence tolerance (used only if strategy='InexactNEwtion' in the call to KINSol):

$$\text{lintol} = (\text{eta} + \text{eps}) * \| \text{fscale} * \mathbf{f}(\mathbf{y}) \|_{\text{L2}}$$
 which is the used to check if the following inequality is satisfied:

$$\| \text{fscale} * (\mathbf{f}(\mathbf{y}) + \mathbf{J}(\mathbf{y}) * \mathbf{p}) \|_{\text{L2}} \leq \text{lintol}$$
 Valid choices are:

$$\begin{aligned} \text{EtaForm} = \text{Type1}, \quad \text{eta} = \frac{\| \mathbf{f}(\mathbf{y}_{(k+1)}) \|_{\text{L2}} - \| \mathbf{f}(\mathbf{y}_k) + \mathbf{J}(\mathbf{y}_k) * \mathbf{p}_k \|_{\text{L2}}}{\| \mathbf{f}(\mathbf{y}_k) \|_{\text{L2}}} \\ & \quad [\| \mathbf{f}(\mathbf{y}_{(k+1)}) \|_{\text{L2}}]^{\alpha} \\ \text{EtaForm} = \text{Type2}, \quad \text{eta} = \text{gamma} * \frac{[\| \mathbf{f}(\mathbf{y}_k) \|_{\text{L2}}]}{[\| \mathbf{f}(\mathbf{y}_{(k+1)}) \|_{\text{L2}}]^{\alpha}} \end{aligned}$$
 EtaForm='Constant'
Eta - constant value for eta [scalar | 0.1]
 Specifies the constant value for eta in the case EtaForm='Constant'.
EtaAlpha - alpha parameter for eta [scalar | 2.0]
 Specifies the parameter alpha in the case EtaForm='Type2'
EtaGamma - gamma parameter for eta [scalar | 0.9]
 Specifies the parameter gamma in the case EtaForm='Type2'
MinBoundEps - lower bound on eps [false | true]
 Specifies whether or not the value of eps is bounded below by 0.01*FuncNormtol.
Constraints - solution constraints [vector]
 Specifies additional constraints on the solution components.

$$\begin{aligned} \text{Constraints}(i) = 0 : & \text{ no constrain on } \mathbf{y}(i) \\ \text{Constraints}(i) = 1 : & \mathbf{y}(i) \geq 0 \\ \text{Constraints}(i) = -1 : & \mathbf{y}(i) \leq 0 \\ \text{Constraints}(i) = 2 : & \mathbf{y}(i) > 0 \\ \text{Constraints}(i) = -2 : & \mathbf{y}(i) < 0 \end{aligned}$$
 If Constraints is not specified, no constraints are applied to y.

LinearSolver - Type of linear solver [Dense | Band | GMRES | BiCGStab | TFQMR]
 Specifies the type of linear solver to be used for the Newton nonlinear solver.
 Valid choices are: Dense (direct, dense Jacobian), GMRES (iterative, scaled preconditioned GMRES), BiCGStab (iterative, scaled preconditioned stabilized BiCG), TFQMR (iterative, scaled preconditioned transpose-free QMR).
 The GMRES, BiCGStab, and TFQMR are matrix-free linear solvers.
JacobianFn - Jacobian function [function]

This property is overloaded. Set this value to a function that returns Jacobian information consistent with the linear solver used (see Linsolver). If not specified, KINSOL uses difference quotient approximations.

For the Dense linear solver, JacobianFn must be of type KINDenseJacFn and must return a dense Jacobian matrix. For the iterative linear solvers, GMRES, BiCGStab, or TFQMR, JacobianFn must be of type KINJactimesVecFn and must return a Jacobian-vector product.

KrylovMaxDim - Maximum number of Krylov subspace vectors [scalar | 10]
 Specifies the maximum number of vectors in the Krylov subspace. This property is used only if an iterative linear solver, GMRES, BiCGStab, or TFQMR is used (see LinSolver).

MaxNumRestarts - Maximum number of GMRES restarts [scalar | 0]
 Specifies the maximum number of times the GMRES (see LinearSolver) solver can be restarted.

PrecModule - Built-in preconditioner module [BBDPre | UserDefined]
 If the PrecModule = 'UserDefined', then the user must provide at least a preconditioner solve function (see PrecSolveFn)
 KINSOL provides a built-in preconditioner module, BBDPre which can only be used with parallel vectors. It provides a preconditioner matrix that is block-diagonal with banded blocks. The blocking corresponds to the distribution of the variable vector among the processors. Each preconditioner block is generated from the Jacobian of the local part (on the current processor) of a given function $g(t,y)$ approximating $f(y)$ (see GlocalFn). The blocks are generated by a difference quotient scheme on each processor independently. This scheme utilizes an assumed banded structure with given half-bandwidths, mldq and mudq (specified through LowerBwidthDQ and UpperBwidthDQ, respectively). However, the banded Jacobian block kept by the scheme has half-bandwidths ml and mu (specified through LowerBwidth and UpperBwidth), which may be smaller.

PrecSetupFn - Preconditioner setup function [function]
 PrecSetupFn specifies an optional function which, together with PrecSolve, defines a right preconditioner matrix which is an approximation to the Newton matrix. PrecSetupFn must be of type KINPrecSetupFn.

PrecSolveFn - Preconditioner solve function [function]
 PrecSolveFn specifies an optional function which must solve a linear system $Pz = r$, for given r . If PrecSolveFn is not defined, no preconditioning will be used. PrecSolveFn must be of type KINPrecSolveFn.

GlocalFn - Local right-hand side approximation function for BBDPre [function]
 If PrecModule is BBDPre, GlocalFn specifies a required function that evaluates a local approximation to the system function. GlocalFn must be of type KINGlocalFn.

GcommFn - Inter-process communication function for BBDPre [function]
 If PrecModule is BBDPre, GcommFn specifies an optional function to perform any inter-process communication required for the evaluation of GlocalFn. GcommFn must be of type KINGcommFn.

LowerBwidth - Jacobian/preconditioner lower bandwidth [scalar | 0]
 This property is overloaded. If the Band linear solver is used (see LinSolver), it specifies the lower half-bandwidth of the band Jacobian approximation. If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used (see LinSolver) and if the BBDPre preconditioner module in KINSOL is used (see PrecModule), it specifies the lower half-bandwidth of the retained banded approximation of the local Jacobian block.
 LowerBwidth defaults to 0 (no sub-diagonals).

UpperBwidth - Jacobian/preconditioner upper bandwidth [scalar | 0]
 This property is overloaded. If the Band linear solver is used (see LinSolver),

it specifies the upper half-bandwidth of the band Jacobian approximation. If one of the three iterative linear solvers, GMRES, BiCGStab, or TFQMR is used (see LinSolver) and if the BBDPre preconditioner module in KINSOL is used (see PrecModule), it specifies the upper half-bandwidth of the retained banded approximation of the local Jacobian block.
 UpperBwidth defaults to 0 (no super-diagonals).

LowerBwidthDQ – BBDPre preconditioner DQ lower bandwidth [scalar | 0]
 Specifies the lower half-bandwidth used in the difference-quotient Jacobian approximation for the BBDPre preconditioner (see PrecModule).

UpperBwidthDQ – BBDPre preconditioner DQ upper bandwidth [scalar | 0]
 Specifies the upper half-bandwidth used in the difference-quotient Jacobian approximation for the BBDPre preconditioner (see PrecModule).

See also

KINDenseJacFn, KINJacTimesVecFn
 KINPrecSetupFn, KINPrecSolveFn
 KINGlocalFn, KINGcommFn

KINMalloc

PURPOSE

KINMalloc allocates and initializes memory for KINSOL.

SYNOPSIS

```
function [] = KINMalloc(fct,n,varargin)
```

DESCRIPTION

KINMalloc allocates and initializes memory for KINSOL.

Usage: KINMalloc (SYSFUN, N [, OPTIONS [, DATA]]);

SYSFUN is a function defining the nonlinear problem $f(y) = 0$.
 This function must return a column vector FY containing the current value of the residual
 N is the (local) problem dimension.
 OPTIONS is an (optional) set of integration options, created with the KINSetOptions function.
 DATA is the (optional) problem data passed unmodified to all user-provided functions when they are called. For example,
 $RES = SYSFUN(Y,DATA)$.

See also: KINSysFn

KINSol

PURPOSE

KINSol solves the nonlinear problem.

SYNOPSIS

```
function [status,y] = KINSol(y0, strategy, yscale, fscale)
```

DESCRIPTION

KINSol solves the nonlinear problem.

Usage: [STATUS, Y] = KINSol(Y0, STRATEGY, YSCALE, FSCALE)

KINSol manages the computational process of computing an approximate solution of the nonlinear system. If the initial guess (initial value assigned to vector Y0) doesn't violate any user-defined constraints, then KINSol attempts to solve the system $f(y)=0$. If an iterative linear solver was specified (see KINSetOptions), KINSol uses a nonlinear Krylov subspace projection method. The Newton-Krylov iterations are stopped if either of the following conditions is satisfied:

$\|f(y)\|_{\text{L-infinity}} <= 0.01 * \text{fnormtol}$

$\|y[i+1] - y[i]\|_{\text{L-infinity}} <= \text{scsteptol}$

However, if the current iterate satisfies the second stopping criterion, it doesn't necessarily mean an approximate solution has been found since the algorithm may have stalled, or the user-specified step tolerance may be too large.

STRATEGY specifies the global strategy applied to the Newton step if it is unsatisfactory. Valid choices are 'None' or 'LineSearch'.

YSCALE is a vector containing diagonal elements of scaling matrix for vector Y chosen so that the components of $\text{YSCALE} \cdot Y$ (as a matrix multiplication) all have about the same magnitude when Y is close to a root of $f(y)$

FSCALE is a vector containing diagonal elements of scaling matrix for $f(y)$ chosen so that the components of $\text{FSCALE} \cdot f(Y)$ (as a matrix multiplication) all have roughly the same magnitude when u is not too near a root of $f(y)$

On return, status is one of the following:

- 0: KINSol succeeded
- 1: The initial y_0 already satisfies the stopping criterion given above
- 2: Stopping tolerance on scaled step length satisfied
- 1: Illegal attempt to call before KINMalloc
- 2: One of the inputs to KINSol is illegal.
- 5: The line search algorithm was unable to find an iterate sufficiently distinct from the current iterate
- 6: The maximum number of nonlinear iterations has been reached
- 7: Five consecutive steps have been taken that satisfy the following inequality:
$$\|\text{yscale} \cdot p\|_{\text{L2}} > 0.99 * \text{mxnewtstep}$$
- 8: The line search algorithm failed to satisfy the beta-condition for too many times.
- 9: The linear solver's solve routine failed in a recoverable manner, but the linear solver is up to date.
- 10: The linear solver's initialization routine failed.
- 11: The linear solver's setup routine failed in an unrecoverable manner.
- 12: The linear solver's solve routine failed in an unrecoverable manner.

See also KINSetOptions, KINGetstats

KINGetStats

PURPOSE

KINGetStats returns statistics for the main KINSOL solver and the linear

SYNOPSIS

```
function si = KINGetStats()
```

DESCRIPTION

KINGetStats returns statistics for the main KINSOL solver and the linear solver used.

Usage: `solver_stats = KINGetStats;`

Fields in the structure `solver_stats`

- o `nfe` - total number evaluations of the nonlinear system function SYSFUN
- o `nni` - total number of nonlinear iterations
- o `nbcf` - total number of beta-condition failures
- o `nbops` - total number of backtrack operations (step length adjustments) performed by the line search algorithm
- o `fnorm` - scaled norm of the nonlinear system function $f(y)$ evaluated at the current iterate: $\|fscale*f(y)\|_L2$
- o `step` - scaled norm (or length) of the step used during the previous iteration: $\|uscale*p\|_L2$
- o `LSInfo` - structure with linear solver statistics

The structure LSinfo has different fields, depending on the linear solver used.

Fields in LSinfo for the 'Dense' linear solver

- o `name` - 'Dense'
- o `njeD` - number of Jacobian evaluations
- o `nfeD` - number of right-hand side function evaluations for difference-quotient Jacobian approximation

Fields in LSinfo for the 'GMRES' or 'BiCGStab' linear solver

- o `name` - 'GMRES' or 'BiCGStab'
- o `nli` - number of linear solver iterations
- o `npe` - number of preconditioner setups
- o `nps` - number of preconditioner solve function calls
- o `ncfl` - number of linear system convergence test failures

KINFree

PURPOSE

KINFree deallocates memory for the KINSOL solver.

SYNOPSIS

```
function [] = KINFree()
```

DESCRIPTION

KINFree deallocates memory for the KINSOL solver.

Usage: KINFree

3.2 Function types

KINDenseJacFn

PURPOSE

KINDenseJacFn - type for user provided dense Jacobian function.

SYNOPSIS

This is a script file.

DESCRIPTION

KINDenseJacFn - type for user provided dense Jacobian function.

The function DJACFUN must be defined as

```
FUNCTION [J, FLAG] = DJACFUN(Y,FY)
```

and must return a matrix J corresponding to the Jacobian of $f(y)$.

The input argument FY contains the current value of $f(y)$.

If a user data structure DATA was specified in KINMalloc, then

DJACFUN must be defined as

```
FUNCTION [J, FLAG, NEW_DATA] = DJACFUN(Y,FY,DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the matrix J and the flag FLAG, the DJACFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function DJACFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also KINSetOptions

NOTE: DJACFUN is specified through the property JacobianFn to KINSetOptions and is used only if the property LinearSolver was set to 'Dense'.

KINBandJacFn

PURPOSE

KINBandJacFn - type for user provided banded Jacobian function.

SYNOPSIS

This is a script file.

DESCRIPTION

KINBandJacFn - type for user provided banded Jacobian function.

The function BJACFUN must be defined as

```
FUNCTION [J, FLAG] = BJACFUN(Y, FY)
```

and must return a matrix J corresponding to the banded Jacobian of f(y).

The input argument FY contains the current value of f(y).

If a user data structure DATA was specified in KINMalloc, then

BJACFUN must be defined as

```
FUNCTION [J, FLAG, NEW_DATA] = BJACFUN(Y, FY, DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the matrix J and the flag FLAG, the BJACFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function BJACFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also KINSetOptions

NOTE: BJACFUN is specified through the property JacobianFn to KINSetOptions and is used only if the property LinearSolver was set to 'Band'.

KINGcommFn

PURPOSE

KINGcommFn - type for user provided communication function (BBDPre).

SYNOPSIS

This is a script file.

DESCRIPTION

KINGcommFn - type for user provided communication function (BBDPre).

The function GCOMFUN must be defined as

```
FUNCTION FLAG = GCOMFUN(Y)
```

and can be used to perform all interprocess communication necessary to evaluate the approximate right-hand side function for the BBDPre preconditioner module.

If a user data structure DATA was specified in KINMalloc, then GCOMFUN must be defined as

```
FUNCTION [FLAG, NEW_DATA] = GCOMFUN(Y, DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then the GCOMFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function GCOMFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also KINGlocalFn, KINSetOptions

NOTES:

GCOMFUN is specified through the GcommFn property in KINSetOptions and is used only if the property PrecModule is set to 'BBDPre'.

Each call to GCOMFUN is preceded by a call to the system function SYSFUN with the same argument Y. Thus GCOMFUN can omit any communication done by SYSFUN if relevant to the evaluation of G by GLOCFUN. If all necessary communication was done by SYSFUN, GCOMFUN need not be provided.

KINGlocalFn

PURPOSE

KINGlocalFn - type for user provided RHS approximation function (BBDPre).

SYNOPSIS

This is a script file.

DESCRIPTION

KINGlocalFn - type for user provided RHS approximation function (BBDPre).

The function GLOCFUN must be defined as

FUNCTION [G, FLAG] = GLOCFUN(Y)

and must return a vector G corresponding to an approximation to f(y) which will be used in the BBDPRE preconditioner module. The case where G is mathematically identical to F is allowed.

If a user data structure DATA was specified in KINMalloc, then GLOCFUN must be defined as

FUNCTION [G, FLAG, NEW_DATA] = GLOCFUN(Y, DATA)

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector G, the GLOCFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function GLOCFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also KINGcommFn, KINSetOptions

NOTE: GLOCFUN is specified through the GlocalFn property in KINSetOptions and is used only if the property PrecModule is set to 'BBDPre'.

KINJacTimesVecFn

PURPOSE

KINJacTimesVecFn - type for user provided Jacobian times vector function.

SYNOPSIS

This is a script file.

DESCRIPTION

KINJacTimesVecFn - type for user provided Jacobian times vector function.

The function JTVFUN must be defined as

```
FUNCTION [JV, NEW_Y, FLAG] = JTVFUN(Y, V, NEW_Y)
```

and must return a vector JV corresponding to the product of the Jacobian of $f(y)$ with the vector v. On input, NEW_Y indicates if the iterate has been updated in the interim. JV must be updated or reevaluated, if appropriate, unless NEW_Y=false. This flag must be reset by the user.

If a user data structure DATA was specified in KINMalloc, then JTVFUN must be defined as

```
FUNCTION [JV, NEW_Y, FLAG, NEW_DATA] = JTVFUN(Y, V, NEW_Y, DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector JV, and flags NEW_Y and FLAG, the JTVFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

If successful, FLAG should be set to 0. If an error occurs, FLAG should be set to a nonzero value.

See also KINSetOptions

NOTE: JTVFUN is specified through the property JacobianFn to KINSetOptions and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.

KINPrecSetupFn

PURPOSE

KINPrecSetupFn - type for user provided preconditioner setup function.

SYNOPSIS

This is a script file.

DESCRIPTION

KINPrecSetupFn - type for user provided preconditioner setup function.

The user-supplied preconditioner setup subroutine should compute the right-preconditioner matrix P used to form the scaled preconditioned linear system:

$$(Df * J(y) * (P^{-1}) * (Dy^{-1})) * (Dy * P * x) = Df * (-F(y))$$

where D_y and D_f denote the diagonal scaling matrices whose diagonal elements are stored in the vectors `YSCALE` and `FSCALE`, respectively.

The preconditioner setup routine (referenced by iterative linear solver modules via `pset` (type `KINSpilsPrecSetupFn`)) will not be called prior to every call made to the `psolve` function, but will instead be called only as often as necessary to achieve convergence of the Newton iteration.

NOTE: If the `PRECSOLVE` function requires no preparation, then a preconditioner setup function need not be given.

The function `PSETFUN` must be defined as

```
FUNCTION FLAG = PSETFUN(Y, YSCALE, FY, FSCALE)
```

The input argument `FY` contains the current value of $f(y)$, while `YSCALE` and `FSCALE` are the scaling vectors for solution and system function, respectively (as passed to `KINSol`)

If a user data structure `DATA` was specified in `KINMalloc`, then `PSETFUN` must be defined as

```
FUNCTION [FLAG, NEW_DATA] = PSETFUN(Y, YSCALE, FY, FSCALE, DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the flag `FLAG`, the `PSETFUN` function must also set `NEW_DATA`. Otherwise, it should set `NEW_DATA= []` (do not set `NEW_DATA = DATA` as it would lead to unnecessary copying).

If successful, `PSETFUN` must return `FLAG=0`. For a recoverable error (in which case the setup will be retried) it must set `FLAG` to a positive integer value. If an unrecoverable error occurs, it must set `FLAG` to a negative value, in which case the solver will halt.

See also `KINPrecSolveFn`, `KINSetOptions`, `KINSol`

NOTE: `PSETFUN` is specified through the property `PrecSetupFn` to `KINSetOptions` and is used only if the property `LinearSolver` was set to '`GMRES`' or '`BiCGStab`'.

KINPrecSolveFn

PURPOSE

`KINPrecSolveFn` - type for user provided preconditioner solve function.

SYNOPSIS

This is a script file.

DESCRIPTION

`KINPrecSolveFn` - type for user provided preconditioner solve function.

The user-supplied preconditioner solve function `PSOLFN` is to solve a linear system $P z = r$ in which the matrix P is

the preconditioner matrix (possibly set implicitly by PSETFUN)

The function PSOLFUN must be defined as

```
FUNCTION [Z, FLAG] = PSOLFUN(Y, YSCALE, FY, FSCALE, R)
```

and must return a vector Z containing the solution of $Pz=r$.

The input argument FY contains the current value of $f(y)$, while YSCALE and FSCALE are the scaling vectors for solution and system function, respectively (as passed to KINSol)

If a user data structure DATA was specified in KINMalloc, then PSOLFUN must be defined as

```
FUNCTION [Z, FLAG, NEW_DATA] = PSOLFUN(Y, YSCALE, FY, FSCALE, R, DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector Z and the flag FLAG, the PSOLFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

If successful, PSOLFUN must return FLAG=0. For a recoverable error it must set FLAG to a positive value (in which case the solver will attempt to correct). If an unrecoverable error occurs, it must set FLAG to a negative value, in which case the solver will halt.

See also KINPrecSetupFn, KINSetOptions

NOTE: PSOLFUN is specified through the property PrecSolveFn to KINSetOptions and is used only if the property LinearSolver was set to 'GMRES' or 'BiCGStab'.

KINSysFn

PURPOSE

KINSysFn - type for user provided system function

SYNOPSIS

This is a script file.

DESCRIPTION

KINSysFn - type for user provided system function

The function SYSFUN must be defined as

```
FUNCTION [FY, FLAG] = SYSFUN(Y)
```

and must return a vector FY corresponding to $f(y)$.

If a user data structure DATA was specified in KINMalloc, then SYSFUN must be defined as

```
FUNCTION [FY, FLAG, NEW_DATA] = SYSFUN(Y, DATA)
```

If the local modifications to the user data structure are needed in other user-provided functions then, besides setting the vector FY, the SYSFUN function must also set NEW_DATA. Otherwise, it should set NEW_DATA=[] (do not set NEW_DATA = DATA as it would lead to unnecessary copying).

The function SYSFUN must set FLAG=0 if successful, FLAG<0 if an unrecoverable failure occurred, or FLAG>0 if a recoverable error occurred.

See also KINMalloc

NOTE: SYSFUN is specified through the KINMalloc function.

4 Supporting modules

This section describes two additional modules in SUNDIALSTB, NVECTOR and PUTILS. The functions in NVECTOR perform various operations on vectors. For serial vectors, all of these operations default to the corresponding MATLAB functions. For parallel vectors, they can be used either on the local portion of the distributed vector or on the global vector (in which case they will trigger an MPI Allreduce operation). The functions in PUTILS are used to run parallel SUNDIALSTB applications. The user should only call the function `mpirun` to launch a parallel MATLAB application. See one of the parallel SUNDIALSTB examples for usage.

The functions in these two additional modules are listed in Table 3 and described in detail in the remainder of this section.

Table 3: The NVECTOR and PUTILS functions

NVECTOR	N_VMax	returns the largest element of x
	N_VMaxNorm	returns the maximum norm of x
	N_VMin	returns the smallest element of x
	N_VDotProd	returns the dot product of two vectors
	N_VWrmsNorm	returns the weighted root mean square norm of x
	N_VWL2Norm	returns the weighted Euclidean L2 norm of x
	N_VL1Norm	returns the L1 norm of x
PUTILS	mpirun mpiruns mpistart	runs parallel examples runs the parallel example on a child MATLAB process lamboot and MPI_Init master (if required)

4.1 NVECTOR functions

N_VDotProd

PURPOSE

N_VDotProd returns the dot product of two vectors

SYNOPSIS

```
function ret = N_VDotProd(x,y,comm)
```

DESCRIPTION

N_VDotProd returns the dot product of two vectors

Usage: RET = N_VDotProd (X, Y [, COMM])

If COMM is not present, N_VDotProd returns the dot product of the local portions of X and Y. Otherwise, it returns the global dot product.

SOURCE CODE

```
1 function ret = N_VDotProd(x,y,comm)
9
10 % Radu Serban <radu@llnl.gov>
11 % Copyright (c) 2005, The Regents of the University of California.
12 % $Revision: 1.1 $Date: 2006/01/06 19:00:10 $
13
14
15 if nargin == 2
16
17     ret = dot(x,y);
18
19 else
20
21     ldot = dot(x,y);
22     gdot = 0.0;
23     MPI_Allreduce(ldot,gdot,'SUM',comm);
24     ret = gdot;
25
26 end
```

N_VL1Norm

PURPOSE

N_VL1Norm returns the L1 norm of x

SYNOPSIS

```
function ret = N_VL1Norm(x,comm)
```

DESCRIPTION

N_VL1Norm returns the L1 norm of x

Usage: RET = N_VL1Norm (X [, COMM])

If COMM is not present, N_VL1Norm returns the L1 norm of the local portion of X. Otherwise, it returns the global L1 norm..

SOURCE CODE

```
1 | function ret = N_VL1Norm(x,comm)
9 |
10| % Radu Serban <radu@llnl.gov>
11| % Copyright (c) 2005, The Regents of the University of California.
12| % $Revision: 1.1 $Date: 2006/01/06 19:00:10 $
13|
14| if nargin == 1
15|
16|     ret = norm(x,1);
17|
18| else
19|
20|     lnrm = norm(x,1);
21|     gnrm = 0.0;
22|     MPI_Allreduce(lnrm,gnrm,'MAX',comm);
23|     ret = gnrm;
24|
25| end
```

N_VMax

PURPOSE

N_VMax returns the largest element of x

SYNOPSIS

```
function ret = N_VMax(x,comm)
```

DESCRIPTION

N_VMax returns the largest element of x

Usage: RET = N_VMax (X [, COMM])

If COMM is not present, N_VMax returns the maximum value of the local portion of X. Otherwise, it returns the global maximum value.

SOURCE CODE

```
1 | function ret = N_VMax(x,comm)
9 |
10| % Radu Serban <radu@llnl.gov>
11| % Copyright (c) 2005, The Regents of the University of California.
12| % $Revision: 1.1 $Date: 2006/01/06 19:00:10 $
```

```

13
14 if nargin == 1
15
16     ret = max(x);
17
18 else
19
20     lmax = max(x);
21     gmax = 0.0;
22     MPI_Allreduce(lmax,gmax,'MAX',comm);
23     ret = gmax;
24
25 end

```

N_VMaxNorm

PURPOSE

`N_VMaxNorm` returns the L-infinity norm of `x`

SYNOPSIS

```
function ret = N_VMaxNorm(x, comm)
```

DESCRIPTION

`N_VMaxNorm` returns the L-infinity norm of `x`

Usage: `RET = N_VMaxNorm (X [, COMM])`

If `COMM` is not present, `N_VMaxNorm` returns the L-infinity norm of the local portion of `X`. Otherwise, it returns the global L-infinity norm..

SOURCE CODE

```

1 function ret = N_VMaxNorm(x, comm)
9
10 % Radu Serban <radu@llnl.gov>
11 % Copyright (c) 2005, The Regents of the University of California.
12 % $Revision: 1.1 $Date: 2006/01/06 19:00:10 $
13
14 if nargin == 1
15
16     ret = norm(x, 'inf');
17
18 else
19
20     lnrm = norm(x, 'inf');
21     gnrm = 0.0;
22     MPI_Allreduce(lnrm,gnrm,'MAX',comm);
23     ret = gnrm;
24
25 end

```

N_VMin

PURPOSE

N_VMin returns the smallest element of x

SYNOPSIS

```
function ret = N_VMin(x,comm)
```

DESCRIPTION

N_VMin returns the smallest element of x

Usage: RET = N_VMin (X [, COMM])

If COMM is not present, N_VMin returns the minimum value of the local portion of X. Otherwise, it returns the global minimum value.

SOURCE CODE

```
1 function ret = N_VMin(x,comm)
2
3 % Radu Serban <radu@llnl.gov>
4 % Copyright (c) 2005, The Regents of the University of California.
5 % $Revision: 1.1 $Date: 2006/01/06 19:00:10 $
6
7 if nargin == 1
8
9     ret = min(x);
10
11 else
12
13     lmin = min(x);
14     gmin = 0.0;
15     MPI_Allreduce(lmin,gmin,'MIN',comm);
16     ret = gmin;
17
18 end
```

N_VWL2Norm

PURPOSE

N_VWL2Norm returns the weighted Euclidean L2 norm of x

SYNOPSIS

```
function ret = N_VWL2Norm(x,w,comm)
```

DESCRIPTION

```

N_VWL2Norm returns the weighted Euclidean L2 norm of x
with weight vector w:
sqrt [(sum (i = 0 to N-1) (x[i]*w[i])^2)]

```

Usage: RET = N_VWL2Norm (X, W [, COMM])

If COMM is not present, N_VWL2Norm returns the weighted L2 norm of the local portion of X. Otherwise, it returns the global weighted L2 norm..

SOURCE CODE

```

1 function ret = N_VWL2Norm(x,w,comm)
11
12 % Radu Serban <radu@llnl.gov>
13 % Copyright (c) 2005, The Regents of the University of California.
14 % $Revision: 1.1 $Date: 2006/01/06 19:00:10 $
15
16 if nargin == 2
17
18     ret = dot(x.^2,w.^2);
19     ret = sqrt(ret);
20
21 else
22
23     lnrm = dot(x.^2,w.^2);
24     gnrm = 0.0;
25     MPI_Allreduce(lnrm,gnrm,'SUM',comm);
26
27     ret = sqrt(gnrm);
28
29 end

```

N_VWrmsNorm

PURPOSE

N_VWrmsNorm returns the weighted root mean square norm of x

SYNOPSIS

```
function ret = N_VWrmsNorm(x,w,comm)
```

DESCRIPTION

N_VWrmsNorm returns the weighted root mean square norm of x
with weight vector w:

sqrt [(sum (i = 0 to N-1) (x[i]*w[i])^2)/N]

Usage: RET = N_VWrmsNorm (X, W [, COMM])

If COMM is not present, N_VWrmsNorm returns the WRMS norm of the local portion of X. Otherwise, it returns the global WRMS norm..

SOURCE CODE

```

1 | function ret = N.VWrmsNorm(x,w,comm)
11 |
12 | % Radu Serban <radu@llnl.gov>
13 | % Copyright (c) 2005, The Regents of the University of California.
14 | % $Revision: 1.1 $Date: 2006/01/06 19:00:11 $
15 |
16 | if nargin == 2
17 |
18 |     ret = dot(x.^2,w.^2);
19 |     ret = sqrt(ret/length(x));
20 |
21 | else
22 |
23 |     lnrm = dot(x.^2,w.^2);
24 |     gnrm = 0.0;
25 |     MPI_Allreduce(lnrm,gnrm,'SUM',comm);
26 |
27 |     ln = length(x);
28 |     gn = 0;
29 |     MPI_Allreduce(ln,gn,'SUM',comm);
30 |
31 |     ret = sqrt(gnrm/gn);
32 |
33 | end

```

4.2 Parallel utilities

mpirun

PURPOSE

MPIRUN runs parallel examples.

SYNOPSIS

```
function [] = mpirun(fct,npe,dbg)
```

DESCRIPTION

MPIRUN runs parallel examples.

Usage: MPIRUN (FCT , NPE [, DBG])

FCT - function to be executed on all MATLAB processes.

NPE - number of processes to be used (including the master).

DBG - flag for debugging [true | false]

If true, spawn MATLAB child processes with a visible xterm.

SOURCE CODE

```
1 function [] = mpirun(fct,npe,dbg)
2
3 % Radu Serban <radu@llnl.gov>
4 % Copyright (c) 2005, The Regents of the University of California.
5 % $Revision: 1.2 $Date: 2006/03/07 01:20:01 $
6
7 ih = isa(fct,'function_handle');
8 is = isa(fct,'char');
9 if ih
10    sh = functions(fct);
11    fct_str = sh.function;
12 elseif is
13    fct_str = fct;
14 else
15    error('mpirun::Unrecognized_function');
16 end
17
18 if exist(fct_str) ~= 2
19    err_msg = sprintf('mpirun::Function %s not in search path.',fct_str);
20    error(err_msg);
21 end
22
23 nslaves = npe-1;
24 mpistart(nslaves);
25
26 debug = false;
27 if (nargin > 2) & dbg
28    debug = true;
29 end
```

```

38 cmd_slaves = sprintf('mpiruns( ''%s '' )',fct_str);
39
40 if debug
41   cmd = 'xterm';
42   args = {'-sb', '-sl', '5000', '-e', 'matlab', '-nosplash', '-nojvm', '-r', cmd_slaves};
43 else
44   cmd = 'matlab';
45   args = {'-nosplash', '-nojvm', '-r', cmd_slaves};
46 end
47
48 [info children errs] = MPI_Comm_spawn(cmd,args,nslaves,'NULL',0,'SELF');
49
50 [info NEWORLD] = MPI_Intercomm_merge(children,0);
51
52 % Put the MPI communicator in the global workspace
53 global sundials_MPI_comm;
54 sundials_MPI_comm = NEWORLD;
55
56 % Get rank of current process and put it in the global workspace
57 [status mype] = MPI_Comm_rank(NEWORLD);
58 global sundials_MPI_rank;
59 sundials_MPI_rank = mype;
60
61 % Call the user main program
62 feval(fct,NEWORLD);
63
64 % Clear the global MPI communicator variable
65 clear sundials_MPI_comm
66

```

mpiruns

PURPOSE

MPIRUNS runs the parallel example on a child MATLAB process.

SYNOPSIS

```
function [] = mpiruns(fct)
```

DESCRIPTION

MPIRUNS runs the parallel example on a child MATLAB process.

Usage: MPIRUNS (FCT)

This function should not be called directly. It is called by `mpirun` on the spawned child processes.

SOURCE CODE

```

1 function [] = mpiruns(fct)
2
3 % Radu Serban <radu@llnl.gov>
4 % Copyright (c) 2005, The Regents of the University of California.
5

```

```

11 % $Revision: 1.2 $Date: 2006/03/07 01:20:01 $
12
13 clc ;
14
15 [dummy hostname]=system( 'hostname ' );
16 fprintf( 'mpiruns::: child MATLAB process on %s\n' ,hostname );
17
18 MPI_Init ;
19
20 MPI_Errhandler_set( 'WORLD' , 'RETURN' );
21
22 [info parent] = MPI_Comm_get_parent ;
23
24 fprintf( 'mpiruns::: waiting to merge MPI intercommunicators ... ' );
25 [info NEWORLD] = MPI_Intercomm_merge( parent ,1 );
26 fprintf( 'OK!\n\n' );
27
28 MPI_Errhandler_set( NEWORLD, 'RETURN' );
29
30 % Put the MPI communicator in the global workspace
31 global sundials_MPI_comm ;
32 sundials_MPI_comm = NEWORLD;
33
34 % Get rank of current process and put it in the global workspace
35 [status mype] = MPI_Comm_rank(NEWORLD);
36 global sundials_MPI_rank ;
37 sundials_MPI_rank = mype;
38
39 fprintf( 'mpiruns::: MPI_rank: %d\n\n' ,mype);
40
41 fprintf( '-----\n\n' );
42
43 % Call the user main program
44 feval( fct ,NEWORLD);
45
46 % Clear the global MPI communicator variable
47 clear sundials_MPI_comm
48
49 % Finalize MPI on this slave
50 MPI_Finalize ;

```

mpistart

PURPOSE

MPISTART invokes lamboot (if required) and MPI_Init (if required).

SYNOPSIS

```
function mpistart(nslaves, rpi, hosts)
```

DESCRIPTION

MPISTART invokes lamboot (if required) and MPI_Init (if required).

Usage: MPISTART [(NSLAVES [, RPI [, HOSTS]])]

MPISTART boots LAM and initializes MPI to match a given number of slave hosts (and rpi) from a given list of hosts. All three args optional.

If they are not defined, HOSTS are taken from a builtin HOSTS list (edit HOSTS at the beginning of this file to match your cluster) or from the bhost file if defined through LAMBHOST (in this order).

If not defined, RPI is taken from the builtin variable RPI (edit it to suit your needs) or from the LAM_MPI_SSI_rpi environment variable (in this order).

SOURCE CODE

```
1 function mpistart(nslaves, rpi, hosts)
16
17 % Heavily based on the LAM_Init function in MPITB.
18
19 %_____
20 % ARGCHECK
21 %_____
22
23 % List of hosts
24
25 if nargin>2
26
27 % Hosts passed as an argument...
28
29 if ~iscell(hosts)
30     error('MPISTART: 3rd arg is not a cell');
31 end
32 for i=1:length(hosts)
33     if ~ischar(hosts{i})
34         error('MPISTART: 3rd arg is not a cell-of-strings');
35     end
36 end
37
38 else
39
40 % Get hosts from file specified in env. var. LAMBHOST
41
42 bfile = getenv('LAMBHOST');
43 if isempty(bfile)
44     error('MPISTART: cannot find list of hosts');
45 end
46 hosts = readHosts(bfile);
47
48 end
49
50 % RPI
51 if nargin>1
```

```

53 % RPI passed as an argument
54
55 if ~ischar(rpi)
56     error('MPISTART: 2nd arg is not a string')
57 end
58
59 % Get full rpi name, if single letter used
60 rpi = rpi_str(rpi);
61 if isempty(rpi)
62     error('MPISTART: 2nd arg is not a known RPI')
63 end
64
65 else
66
67 % Get RPI from env. var. LAM_MPI_SSI_rpi
68
69 RPI = getenv('LAM_MPI_SSI_rpi');
70 if isempty(RPI)
71 % If LAM_MPI_SSI_rpi not defined, use RPI='tcp'
72     RPI = 'tcp';
73 end
74 rpi = rpi_str(RPI);
75
76 end
77
78 % Number of slaves
79
80 if nargin>0
81     if ~isreal(nslaves) || fix(nslaves) ~= nslaves || nslaves >= length(hosts)
82         error('MPISTART: 1st arg is not a valid #slaves')
83     end
84 else
85     nslaves = length(hosts)-1;
86 end
87
88 %_____
89 % LAMHALT %
90 %_____
91 % reasons to lamhalt:
92 % - not enough nodes (nslv+1) % NHL < NSLAVES+1
93 % - localhost not in list % weird - just lamboot (NHL=0)
94 % - localhost not last in list % weird - just lamboot (NHL=0)
95 %
96
97 % Lam Nodes Output
98 [stat, LNO] = system('lamnodes');
99 if ~stat
100
101     emptyflag = false;
102     if isempty(LNO)
103         % this shouldn't happen
104         emptyflag=true;
105         % it's MATLAB's fault I think
106         fprintf('pushing stubborn MATLAB system call (lamnodes): ');

```

```

107 end
108
109 while isempty(LNO) || stat
110   fprintf('.');
111   [stat, LNO] = system('lamnodes');
112 end
113 if emptyflag
114   fprintf('\n');
115 end
116
117 LF = char(10);
118 LNO = split(LNO,LF); % split lines in rows at \n
119
120 [stat, NHL] = system('lamnodes|wc-1'); % Number of Hosts in Lamnodes
121
122 emptyflag = false; % again,
123 if isempty(NHL) % this shouldn't happen
124   emptyflag=true; % it's MATLAB's fault I think
125   fprintf('pushing stubborn MATLAB system' 'call(lamnodes|wc):');
126 end
127 while isempty(NHL) || stat
128   fprintf('.');
129   [stat, NHL] = system('lamnodes|wc-1');
130 end
131 if emptyflag
132   fprintf('\n');
133 end
134
135 NHL = str2num(NHL);
136 if NHL ~= size(LNO,1) || ~ NHL>0 % Oh my, logic error
137   NHL= 0; % pretend there are no nodes
138   disp('MPISTART: internal logic error: lamboot')
139 end % to force lamboot w/o lamhalt
140 if isempty(findstr(LNO(end,:), 'this_node')) % master computer last in list
141   disp('MPISTART: local host is not last in nodelist, hope that''s right')
142 beforeflag=0;
143 for i=1:size(LNO,1)
144   if ~isempty(findstr(LNO(i,:), 'this_node'))
145     beforeflag=1;
146     break; % well, not 1st but it's there
147   end
148 end % we already warned the user
149 if ~beforeflag % Oh my, incredible, not there
150   NHL= 0; % pretend there are no nodes
151   disp('MPISTART: local host not in LAM? lamboot')
152 end
153 end % to force lamboot w/o lamhalt
154
155 if NHL > 0 % accurately account multiprocessors
156   NCL = 0; % number of CPUs in lamnodes
157   for i=1:size(LNO,1) % add the 2nd ":"-separated
158     fields=split(LNO(i,:), ':'); % field, ie, #CPUs
159     NCL = NCL + str2num(fields(2,:));
160   end

```

```

161 if NCL<NHL                                % Oh my, logic error
162   NHL= 0;                                     % pretend there are no nodes
163   disp( 'MPISTART:_internal_logic_error:_lamboot' )
164 else
165   % update count
166   NHL=NCL;
167 end                                           % can't get count from MPI,
168 end                                           % since might be not _Init'ed

169 if NHL < nslaves+1                           % we have to lamboot

170
171 % but avoid getting caught
172 [infI flgI]=MPI_Initialized;                  % Init?
173 [infF flgF]=MPI_Finalized;                    % Finalize?
174 if infI || infF
175   error( 'MPISTART:_error_calling:_Initialized/_Finalized?' )
176 end
177 if flgI && ~flgF                           % avoid hangup due to
178   MPI_Finalize;                             % imminent lamhalt
179   clear MPI_*
180   disp( 'MPISTART:_MPI_already_used_-clearing_-before_-lamboot' )
181 end                                           % force MPI_Init in Mast/Ping
182                                           % by pretending "not _Init"
183 if NHL > 0                                    % avoid lamhalt in weird cases
184   disp( 'MPISTART:_halting_LAM' )
185   system( 'lamhalt' );
186 end                                           % won't get caught on this
187 end
188
189 %
190 % LAMBOOT
191 %
192 %
193 % reasons to lamboot:                         %
194 % - not lambooted yet                         % stat~=0
195 % - lamhalted above (or weird) % NHL < NSLAVES+1 (0 _is_ <)
196 %

197 if stat || NHL<nslaves+1

198 HNAMS=hosts{end};
199 for i=nslaves:-1:1
200   HNAMS=strvcat( hosts{ i } ,HNAMS);
201 end
202 HNAMS = HNAMS';                               % transpose for "for"

203 fid=fopen( 'bhost' , 'wt' );
204 for h = HNAMS
205   fprintf(fid , '%s\n',h');                   % write slaves' hostnames
206 end
207 fclose(fid);
208 disp( 'MPISTART:_booting_LAM' )

209 stat = system( 'lamboot -s -v bhost' );
210
211
212
213
214

```

```

215 if stat                                     % again , this shouldn't happen
216   fprintf( 'pushing \stubborn MATLAB" system" \call \lamboot:\n' );
217   while stat
218     fprintf( '.' ); stat = system( 'lamboot -s -v -bhost' );
219   end
220   fprintf( '\n' );
221 end
222
223 system( 'rm -f bhost' );                   % don't need bhost anymore
224 end                                         % won't wipe on exit/could lamhalt
225
226 %
227 % RPI CHECK
228 %
229
230 [infI flgI] = MPI_Initialized;             % Init?
231 [infF flgF] = MPI_Finalized;               % Finalize?
232
233 if infI || infF
234   error( 'MPISTART: \error \calling \ Initialized / \Finalized?' )
235 end
236
237 if flgI && ~flgF                         % Perfect , ready to start
238 else                                         % something we could fix?
239   if flgI || flgF
240     clear MPI_*
241     disp( 'MPISTART: \MPI\already \used - \clearing' ) % must start over
242   end
243
244 MPI_Init;
245 end
246
247 %
248 % NSLAVES CHECK
249 %
250
251 [info attr flag] = MPI_Attr_get( MPI_COMM_WORLD, MPI_UNIVERSE_SIZE );
252 if info | ~flag
253   error( 'MPISTART: \attribute \MPI_UNIVERSE_SIZE \does \not \exist?' )
254 end
255 if attr<2
256   error( 'MPISTART: \required \2 \computers \in \LAM' )
257 end
258
259 %
260
261 function hosts = readHosts( bfile )
262 hosts = [];
263
264 fid = fopen( bfile );
265 if fid == -1
266   fprintf( 'Cannot \open \bhost \file \s\n' , bfile );
267   return;

```

```

269 end
270
271 i = 0;
272 while ~feof(fid)
273 % get a line
274 l = fgetl(fid);
275 % Discard comments
276 ic = min(strfind(l, '#'));
277 if ~isempty(ic), l = l(1:ic-1); end
278 % Test if there is anything left :-
279 if isempty(l), continue; end
280 % Got a new host
281 i = i + 1;
282 % Stop at first blank or tab=char(9)
283 indx = find((l==' ') | (l==char(9)));
284 if isempty(indx)
285 hosts{i} = 1;
286 else
287 hosts{i} = l(1:min(indx));
288 end
289 end
290
291 fclose(fid);
292
293 %
294 %=====
295
296 function rpi = rpi_str(c)
297 %RPLSTR Full LAM SSI RPI string given initial letter(s)
298 %
299 % rpi = rpi_str (c)
300 %
301 % c initial char(s) of rpi name: t,l,u,s
302 % rpi full rpi name, one of: tcp , lamd , usysv , sysv
303 % Use '' if c doesn't match to any supported rpi
304 %
305
306 flag = nargin~=1 || isempty(c) || ~ischar(c);
307 if flag
308 return
309 end
310
311 c=lower(c(1));
312 rpis={'tcp','lamd','usysv','sysv','none'}; % 'none' is sentinel
313
314 for i=1:length(rpis)
315 if rpis{i}(1)==c
316 break
317 end
318 end
319
320 if i<length(rpis) % normal cases
321 rpi=rpis{i};
322 else

```

```
323 |   rpi='';  
324 | end                                % no way, unknown rpi
```

References

- [1] A. M. Collier, A. C. Hindmarsh, R. Serban, and C.S. Woodward. User Documentation for KINSOL v2.2.0. Technical Report UCRL-SM-208116, LLNL, 2004.
- [2] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS, suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, (in press), 2004.
- [3] A. C. Hindmarsh and R. Serban. User Documentation for CVODES v2.1.0. Technical report, LLNL, 2004. UCRL-SM-208111.